COMPUTABLE ANALYSIS AND DIGITAL SIGNAL PROCESSING

by
P. S. PUROHIT

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY KANPUR
AUGUST, 1985

COMPUTABLE ANALYSIS AND DIGITAL SIGNAL PROCESSING

A Thesis Submitted
In Partial Fulfilment of the Requirements
for the Degree of

MASTER OF TECHNOLOGY

by
P S PUROHIT

to the

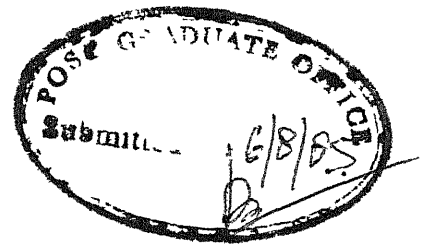
DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY KANPUR
AUGUST, 1985

EE-1985-M-PUR-COM.

-7 8 €

91867

To My Parents

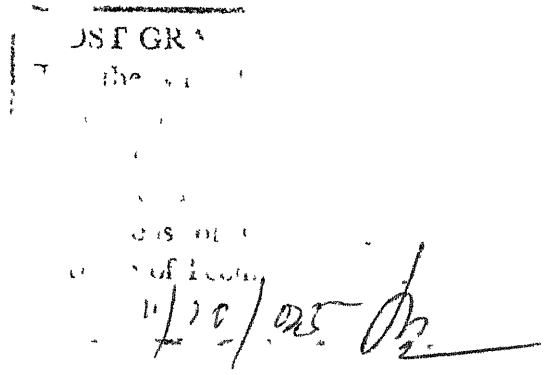


CERTIFICATE

This is to certify that the work on 'COMPUTABLE ANALYSIS AND DIGITAL SIGNAL PROCESSING' by Mr. P.S. Purohit has been carried out under my supervision and this has not been submitted elsewhere for a degree.

(V.P. SINHA)
Professor
Department of Electrical Engineering
Indian Institute of Technology
Kanpur - 208016

August, 1985.



Acknowledgement

It is with a deep sense of gratitude that I express my indebtedness to my thesis supervisor Dr. V.P. Sinha, who apart from providing invaluable guidance in this thesis work, was also a source of inspiration throughout my stay in the campus.

I am also thankful to my friends especially E.G. Rajan, P.G. Poonacha, V.A. Topkar and Siya Ram for the fruitful and interesting discussions throughout the tenure of my work.

Finally, I thank Mr. Yogendra for his skilful typing.

August, 1985. Prithvi Singh Purohit

ABSTRACT

The structure of real numbers is too rich to admit their proper representation even on idealized computer such as an Unlimited Register Machine. It follows that if a physical problem is formulated in terms of real analysis, then there would be an unbridgable gap between what analysis would show about the nature of the physical problem and what a computer simulation would yield about it. An attractive alternative is to switch over from real numbers to what are called computable numbers. The analysis based on computable numbers centers around the fact that a number exists if and only if it can be computed.

In this thesis an introductory account of computable numbers and computable analysis is presented for the readers concerned with digital signal processing. This is done with a view to see how far it would be feasible and worthwhile to reformulate concepts and algorithms of digital signal processing in terms of computable analysis.

Contents

Section 1

Introduction

1.1 Motivation - Real analysis v/s computable analysis :

In trying to introduce the notion of computable numbers and explain what purpose is served by them, it is perhaps best to start with the example given by Aberth [1] .

Let a function be defined as follows :

$f(x) = 0$ if x is a rational number

$f(x) = 1$ if x is an irrational number

Now let us see, what the value f(x) assumes, when x equals the constant γ, where γ is the Eulerb constant and defined as

$$\gamma = \lim_{n} \left(\sum_{j=1}^{n} \frac{1}{j} - \ln n \right)$$

At the present time it is not known whether γ is rational or irrational, so we cannot assign a value to $f(\gamma)$.

In real analysis this difficulty is commonly explained away by saying that while the function does exist.

Our not being able to compute its value simply reflects the imperfect state of our knowledge.

But then what does it really mean to say that something does exist but we cannot exactly ascertain what it is. One could take the view that a number exists only to the extent that we can compute it. It is this view which has led to the notion of computable numbers, functions or sequence. Detailed description about computable analysis and computable number will be given in section 4, but here we shall see what we mean by a computable number in very broad terms.

Let us begin by considering a simple example illustrating the fact that if we were to consider computation in traditional manner we would perhaps look at series expansion

$$e^x = 1 + x + \frac{x^2}{\underline{|2}} + \frac{x^3}{\underline{|3}} + \cdots$$

and say that determining its value is only a partial sum of the series. But then this leaves an entirely open question about errors. If we wish to incorporate here the fact that

its computation be possible using a finite programme in a finite number of steps, it is not enough to just give the series expansion. What we need in addition is to specify on error bound such that it can be ascertained within this error bound by an algorithm. We will use X(s) to denote the computable numbers approximation of X which has an error of $\pm \frac{1}{s}$. As we shall see, X(s) is a number that can be determined by computation on an idealized computer and the set of values X(s) for different values of 's' as the outputs of the same computer program producing these values for different values of the parameter s.

As we will see later also in detail a computable number can be taken in general to be a programme or algorithm of the kind that can compute a function within the required error bound and terminate execution after finite computations.

Now to illustrate the difference between computable analysis and real analysis, we will consider two elementary results of analysis :

1. The first one is the Rolle's theorem, It states that if f(x) is a continuous function on a closed interval a,b with f(a) and f(b) of opposite signs, there is at least one real number C in a,b such that f(c) = 0. Now if we

switch over to computable numbers then it can be shown that this theorem does not hold. More precisely it can be shown that if f(x) is a computable function on the closed interval a,b with f(a) and f(b) of opposite signs, then there may not be a computable number C in a,b such that f(c) = 0 [8].

2. The second result is that function f(x) continuous on a closed interval a,b assumes its supremum at a point in this interval. The corresponding computable analysis result is that a computable function f(x) continuous on a closed interval a,b may not assume its supremum at a point in this interval.

Thus there are areas of differences between computable and real analysis.

1.2 Digital Signal Processing

Signal processing is a field in which we use programs to process the raw data or for designing digital signal processing hardware systems. Theoretical results in digital signal processing are primarily based on real analysis. But when models based on real analysis have to be implemented on computing machines, there are serious problems owing to limitations of machines in representing real numbers. Thus there

a ground to think that if one uses computable analysis, it can help evolve a more realistic theoretical framework for digical signal processing because computable numbers admit exact representation on idealized computer.

The aim of this thesis is to bring together those basic concepts and results of computer analysis in an introductory fashion that are of potential use in digital signal processing. The original plan was to carry out reformulation of the theory of digital filters in terms of computable numbers, however, as work progressed, it was felt that the task was not as straightforward. There were independent areas of work connected with computable analysis that needed to be studied and sorted out. The work reported here is the result of such a study.

3.3 Overview of Sections

As the aims of thesis are to represent computable number concept and connect Digital Signal Processing with it, sections 2 to 4 have discussions centred around computable number, computable analysis and computable arithmatic. Section 2 explains the need of computable numbers in place of real numbers for computational work.

Section 3 is concerned with general discussion on computable analysis. Here the so called most accurate possible (MAP) real analysis is compared with computable analysis. A brief historical account is also given.

Section 4 is about computable number arithematic operations and relations with computable numbers are explained and the fact that computable numbers form a field is discussed in detail.

Section 5 and 6 explain the use of computable numbers in Digital Signal Processing. Section 5 contains an account of the modification required to make computable analysis a practical analysis and attempt is also made to show the possible algorithmic approaches to implement the required modifications.

Section 6 deals with the digital signal processing. An effort has been made to reformulate the basic properties of a DSP system in terms of computable numbers. The Dead band effect and problem of sensitivity are considered.

Section 7 contains, in addition to concluding reamarks, certain suggestions for further work.

Section 2

Appropriateness of Computable Numbers

2.1 The Excessive Richness of Real Numbers

In the traditional hierarchy of numbers we start with natural numbers followed by integers, rationals and ultimately arrive at the real numbers. In the real number system we reach a state of prefection which cannot be excelled so to say. Among the various mathematical entities that we know so far, real numbers have the richest structure possible. Further, for purposes of modelling and analysing physical systems, the real number system furnishes a foundation which is well tested and of long standing.

But this enormous richness of the structure of the real number system is in some ways a hindrance to a realistic modelling of physical processes and systems. In the early days of scientific development when the scale of physical problems was not so large and when analytical models and solutions were considered adequate, this fact did not draw much attention.

In the modern setting, however, when we wish to model, represent and analyze very large systems purely by computational means, the use of the real number system and

mathematical analysis based on it cannot be accepted without reservations. There are two very basic issues involved here.

Firstly the concept of real number as we commonly understand it todays does not contain within it any reference to computation. We are concerned with the fact that it exists, but we are not concerned whether there is a particular algorithm or construction by which its value can be determined with a given accuracy and with a finite amount of computational effort. Yet when our objective is to simulate mathematical models of physical processes computationally, then one cannot meaningfully conceive of a number except in terms of an algorithm that produces it. There is thus the basic limitation with real numbers that there may be those that cannot be defined in terms of effective computational algorithms. In case it is so, then there would remain on ubridgable gap between the mathematical models and the computational simulations for them.

Secondly, large scale computations require the use of computers which when reduced to their essentials, are finite state machines constrained by limitations of memory and speed in a manner that does not permit an exact representation of computations with reals.

Traditionally, these limitations are handled by using what is known as most accurate possible (MAP) representation of

real number [7] . In this representation, a real number is represented as accurately as possible on a given computer and the computations are carried out maintaining the highest accuracy possible. It is hoped that end result would this way automatically have the necessary accuracies. A question that arises is the following :

Instead of 'fitting' the real number system on to a computing machine can we not work backwards. That is, starting with the structure of computations as they are carried out on a computer, can we choose a number system that is exactly implementable atleast on an 'Idealized' machine and is also close enough to the real number system.

In regard to this question, we find from the literature [1] that the computable number system is one such choice. In this number system every real number is approximated by computable values lying within the errorbound specified in advance. This system acknowledges the limitations of the computer in representation of real numbers and treats real numbers as knowable only to within certain finite errors by means of computational algorithms.

In order to fully appreciate the significant of computable numbers, it is perhaps essential first to have an

idea of Idealized computing machine models and their limitations, and to see how computable numbers serve as a matching device between computers and the real number system.

2.2 Idealized Computer Models

In the literature, ideal computer models range from Turing machines to some higher level models as Unlimited Register Machine [2] . All these models rely on the availability of infinite memory in one way or the other. On the ground of the thesis [2] that all these de different models are computationally equivalent, we propose to confine ourselves to the URM model.

The URM model consists of an infinite number of registers labelled R1, R2, R3, ... etc., each of which contains a natural number. This looks as shown in fig. (2.1)

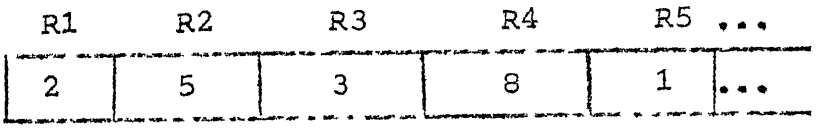


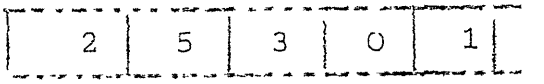
Fig. 2.1 Representation of a URM

For computations on URM, one has to write a program containing instructions from a set of standard instructions

which a URM can execute. The standard instructions are :

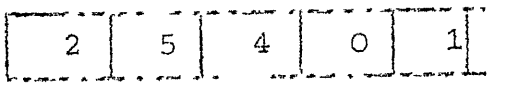
1. Zero Instruction : By executing this instruction URM puts '0' in the nth register. It is denoted by Z(n) for n = 1,2,3,... etc.

Example Z (4)



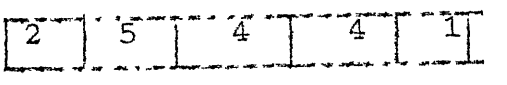
2. Successor Instruction : This instruction increases the number contained in R_n by 1 and is represented as S(n) for n = 1, 2, 3 ... etc.

Example : S(3)



3. Transfer Instruction : This instruction, represented as T(m,n), replaces the contents of R_n by that of R_m, for m, n = 1,2,3, ... etc.

Example : T(3,4)

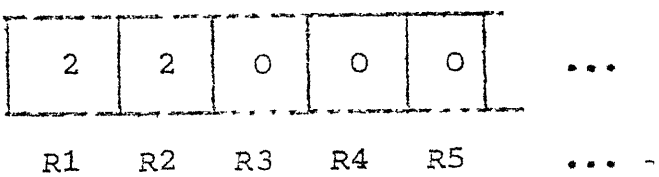


4. Jump Instruction : This instruction branches off execution to the p^{th} instruction if contents of R_m equal the contents of R_n and is represented as J(m,n,p) where m,n,p = 1,2,3, ... etc.

Example J(4,6,20) - if contents of R_4 = contents of R_6
then go to instruction no. 20,
otherwise go to next instruction.

An example of a URM machine programme for computing $(2)^2$ is the following :

```
 1  Z(4)
 2  Z(5)
 3  Z(3)
 4  S(3)
 5  S(4)
 6  J(3,2,8)
 7  J(1,1,4)
 8  S(5)
 9  J(5,1,11)
10  J(1,1,3)
```



This example shows how a function can be represented by a program of finite instructions.

At this point a word about the provision of unlimited memory in the URM model would be appropriate. If we assume that one instruction can be put in one memory register then a program in general can be written in a finite number of memory locations. But there is no upper limit to the number of locations that may be utilized in a particular program. The unlimited register model is assumed to ensure that any finite step program, no matter how large, can be accomodated in the memory.

Another assumption made for the URM model is that the word length of the computer be unlimited. This again is to ensure that integers, no matter how large, can be stored. After examining this idealized computer model, we now turn to computable number and real number representations and their suitability.

2.3 Representing Numbers on Computers

About the idealized computer model, a natural question to ask is whether an exact representation of numbers system is possible on it. If it so turns out that a given number system cannot be exactly represented even on an idealized model then the question of being able to represent it on a practical machine does not arise at all. The point that goes strongly in favour of computable numbers is that while not all real numbers have an exact representation on a URM or any equivalent idealized machine, all computable number do have such representations by definition. Computable numbers, as we will see in Section 4, by their very definition consist of programmes on a URM. Two specific properties of computable numbers that are of fundamental significance in this context are the following.

1. A computable number consists of an effective algorithm that can be realized as a programme on a URM.

2. Computable numbers like the real numbers have the structure of a field, a fact that is known as Rice theorem.

In the light of these two facts it is reasonable to say that computable numbers provide theoretical basis for computer based system studies that is more sound than the traditional one based on real numbers.

2.4 Computable Number - Recursive or Constructive :

Before we close this general discussion about the appropriateness of computable numbers, it is worth while examining their nature in terms of the notions of computability and constructibility. This discussion is based on the idea that all recursive numbers are not constructive [3].

Here 'recursive' means while constructing the number, a computer computes its first decimal digit then the next and so on. The programme also has some condition to terminate the execution. But it is not certain that the condition will ever be met. The term 'Constructive' means that the condition to terminate the execution of programme will be met after a finite number of computations. Myhill has given examples of two recursive number sets that are not constructive and one that is also constructive.

The first one is 'R_d', the set of decimal expansion of real numbers. Such the decimal expanded form terminates at the n^{th} decimal where 'n' is the 'critical number of π'. Here the critical number of π means the number places after which the string 7777 appears in the decimal expansion of π. So far it is not known what this number is.

To give a specific example, let us construct a number, x, from the set R_d, as follows, we put x = 0.3333 ... , and continue repeating 3 till we reach an odd place 2n+1 such that 2n+1 is the'critical number of π', then put 4 at $2n+1^{st}$ place and repeat the process.

The second one is R_1, the set of all 'locatable' real numbers. A number of this set can be constructed as follows.

Consider a real number x and an integer B serving as the upperbound limit. Then compare x with -B, -B+1, ... 0, ... B-1, B. Suppose we find by the comparison a number k such that B+k is less than x and B+k+1 is greater than x. Let x be a number whose integer is B+k. For the decimal part, compare x with $B+k+\frac{n}{10}$ for n = 1,2,3 ... and find s such that $B+k+\frac{s}{10} < x < B+k+\frac{s+1}{10}$. The decimal place of x is then occupied by s. Next we carry out similar comparison with $B+k+\frac{s}{10}$ in place of B+k.

This gives the second place of decimal. Repeating this defines the infinite decimal expansion of x.

The third one is R_{da} the set of decimally approximate real numbers and whose elements are such that for any given rational $\varepsilon > 0$ we can find a finite decimal expansion d such that $x - d < \varepsilon$ then $d \in R_{da}$.

Myhill has given theorems stating that R_1 and R_d are not closed under addition and R_{da} forms a field. Thus decimally approximated real numbers are likely candidates for a constructive analysis but not R_1 or R_d. The set R_{da}, R_1 and R_d are all recursive, but R_{da} is the only one that is constructive as well. Based on this observation of Myhills, we can say that in working with the set R_{da} we are dealing with computable processes, where the error ε of R_{da} corresponds to the error $\frac{1}{s}$ of a computable number $\mathbf{X}(s)$. On further comparison we find that the field R_{da} is the same as the field of computable numbers. We thus conclude following Myhill that computable numbers are recursive as well as constructive in the sense defined above. Now we can enumerate a few examples of computable numbers which will strengthen the discussion.

2.5 A Few Examples of Computable Numbers

To fix the idea discussed, so far we list here two examples of computable numbers. We will see from these examples that while the set of computable numbers is smaller than the set of reals, the computable number set includes all numbers of practical interest.

1. All rationals are computable numbers.
2. Two computable numbers when added, subtracted, multiplied or divided again yield a computable number.

Section 3

About Computable Analysis

3.1.1 Introductions

Having seen broadly what computable numbers are and what their place is in relation to the classical number systems, the next natural step is to examine what is meant by computable analysis.

As real analysis in its classical form does not take into account the limitations in computations on a machine, we have two alternative courses as described in the previous section, which we can summarize as follows :

1. Real analysis be used but real number representation on machine in the MAP form on that machine.
2. We resort to a representation of numbers which is matched to suit the limitations of the machine on one hand and has all the properties to admit a complete analysis on that number system analogous to real analysis.

What we get persuing the first alternative is the so called Most Accurate Possible (MAP) real analysis, and the second

alternative consists of computable analysis. Here we will compare these two approaches and will give our arguments why computable analysis be favoured.

We will also point out the interconnections of computable analysis with developments in the areas of constructive analysis and interval analysis. It is our thinking that studying these connections is important from the point of view of making computable analysis practicable. It will also help in particular, in clarifying the fundamental fact underlying computable analysis that a number is a program.

3.1.2 Difficulties with MAP real Analysis

To begin with, let us concentrate on what are the difficulties which make MAP real analysis inferior to computable analysis. MAP real analysis has the following four well known pitfalls.

1. Real number is represented with some approximation because a machine cannot represent an infinite decimal real number value. So an error exists at each computation and this error part goes on building up as computations continue.

Let us say for instance we wish to compute

$$F(x) = x - \frac{x^3}{\lfloor 3} + \frac{x^5}{\lfloor 5} + \frac{x^{2n-1}}{\lfloor 2n-1}$$

Let y be the MAP representation of x, such that

$$y - \varepsilon < x < y + \varepsilon$$

where ε is the error bound, then in computation for F(x) in MAP real analysis first term introduces the error $\pm$. Second term $\frac{x^3}{\lfloor 3}$ has error bound

$$\varepsilon_2 = \frac{(y \pm \varepsilon)^3}{\lfloor 3} - \frac{y^3}{\lfloor 3}$$

$$= \frac{1}{6}(3y^2\varepsilon + 3y\varepsilon^2 + \varepsilon^3)$$

$$\varepsilon_{sum(2)} = \varepsilon_1 + \varepsilon_2 = \varepsilon + \frac{1}{6}(3y^2\varepsilon + 3y\varepsilon^2 + \varepsilon^3)$$

This way we may compute the summation of n terms and the corresponding error $\varepsilon_{sum(n)}$ goes on increasing with n. Following the MAP approach there is no way for us to exert any meaningful control on this error.

2. We obtain two distinct results for the same function when it is computed on two different computing machines, e.g., a hand claculator and a 20 decimal place precision computer.

3. Since MAP real analysis is not error conscious, certain results may be obtained which are totally unacceptable.

Let us compute $\frac{1}{5(1 + 10^{-23}) - 5}$ using

(a) ten decimal precision (b) 30 decimal precision

Solving with ten decimal precision

$$X = \frac{1}{5(1 + 10^{-23}) - 5}$$

$$= \frac{1}{5(0.9999999999 - 1)}$$

$$= -\frac{1}{5 \times 10^{-10}} = -\frac{1}{5} \times 10^{10} \qquad \text{-- (A)}$$

Now calculation using thirty decimal precision

$$X = \frac{1}{5(1 + 10^{-23}) - 5} = \frac{1}{5 \times 10^{-23}}$$

$$= +\frac{1}{5} \times 10^{23} \qquad \text{-- (B)}$$

We see that results (A) and (B) are too divergent.

4. One refinement of MAP analysis is interval analysis, in which in addition to the MAP computation an error analysis is simultaneously carried out. Even in this refined form, one has no way of controlling error to the required value.

All these four difficulties with the MAP real analysis, are a result of the fact that there are errors in computation and nothing can be said in advance about their values. One has to just compute to the highest accuracy possible and hope for the best. If we wish to improve the situation then we have to look for some form of analysis which involves the concept of error bound in the number representation itself. Efforts in this direction have given rise to what is known as computable analysis.

3.1.3 Getting over the Difficulties

As we shall see in section 4, a computable number is always thought of in terms of an error bound as a parameter. Thus we speak of a computable number X(s), which represents the real value X, where the value of X lies in the range $X(s) - \frac{1}{s}$ to $X(s) + \frac{1}{s}$. If we think of numbers in this manner, then difficulties faced in MAP real analysis can be overcome one by one.

Difficulty (1) with MAP real analysis is removed as the final error bound on the result is fixed ($\pm \frac{1}{s}$ here) before we start execution of the program. So the error build up owing to large number of computations will not be more than the required error bound. Thus if we wish to compute

$$X(100) = \text{Sin } 100^{\circ}$$

By the summation of series expansion of Sin 100^{o}, is to be continued upto the point that the truncation error is less than or equal to $\frac{1}{100}$. Computable number concept thus ensures a fixed error bound on the result.

Difficulty (2) is due to different precisions of different computing machines. But a computable number computed on two different machines will have the same value because both the computable numbers will have equal error approximation to a real number.

$$y(100) = \text{Sin } 20^{o} * \log 5.31$$

represented computable analysis version of the real number

$$y = \text{Sin } 20^{o} * \log 5.31$$

Value y(100), when computed on one machine having 10 decimal precision will have $\pm \frac{1}{100}$ error bound. On the other machine having 20 decimal precision computable number y(100) will have $\pm \frac{1}{100}$ error bound. In Section 4, we will see that two computable number are called equal if their graphical representations have a common overlapping region. Here value y has to be in both the computed numbers. Thus we can say that both machines will give the same computable number.

Difficulty (3) is that sometimes MAP real analysis results are totally unacceptable. In computable analysis, the

machine will give a result with the required error bound, although this may require many more computations than MAP. If result cannot be obtained to within required error bounds, in the programme provision can be made to indicate the situation. Thus there is no chance of unacceptable results going unnoticed.

Difficulty (4) is lack of control on error in MAP real analysis cannot serve our purpose because there is no way of computing result with the required error bound. Computable numbers by definition assure about that.

3.2 A Sub-analysis of Real Analysis

To understand the difference between MAP real analysis and computable analysis further, it is useful to ask the question 'Is computable analysis a sub-analysis of Real Analysis?' There are two ways to look at the relationship between real analysis and computable analysis.

One way is to start from the fact that computable numbers are real numbers but not all real numbers are computable numbers. Yet, a good body of results of classical real analysis holds for computable number too. In this sense one can say that the computable analysis is a sub-analysis of real analysis.

Another way is to include in the number representation the error bound, within which a number is known. This is the computable number concept. Real numbers can be taken as a limiting case of computable numbers with error bound reduced to zero. In this sense one can say that real analysis as a subanalysis of computable analysis. Following Aberth we prefer the first approach.

3.3 Basic Interconnections

Exploring differences between MAP real analysis and computable analysis we can recall various interconnections of these both with error analysis and numerical analysis. An overall idea of these interconnections is perhaps best had from the diagram shown in figure . As shown there, the MAP real analysis does not provide error information but only the value. Interval analysis is the name of the analysis which carries MAP real analysis and error bound computation both while computing. Computable analysis goes one step ahead by regulating maximum error to the required value by proper feedback procedure.

3.4 Historical Comments

The full picture about computable analysis would not be complete unless we take into account certain concurrent

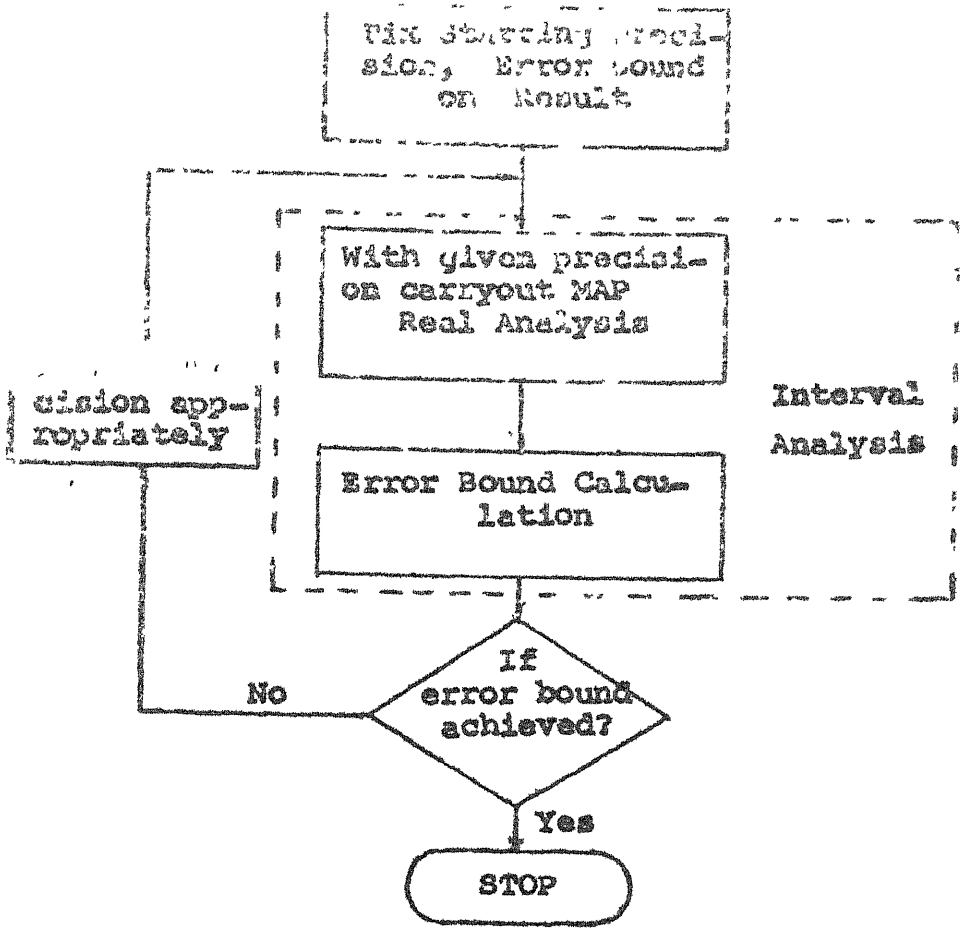


Fig. 3.1(a) COMPUTABLE ANALYSIS

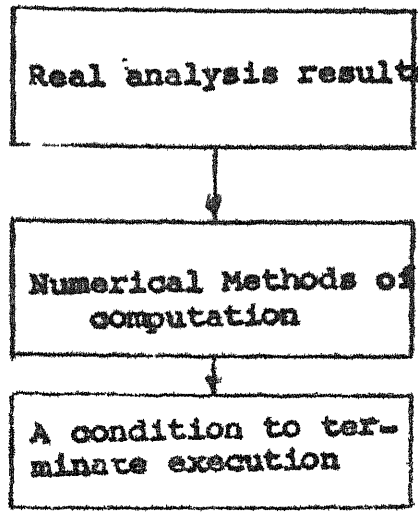


Fig. 3.1(b) MAP REAL ANALYSIS

developments in constructive analysis and interval analysis.

3.4.1 Constructive Analysis : Constructive analysis can be said to have emerged in concrete form from the work of Bishop . In his book 'Foundation of Constructive Analysis' he carried out a reformulation of real analysis in terms of the concept of functions and numbers that can be computed. A more recent book on this analysis is by Bridges. Stress in constructive analysis is on what the results of important real analysis theorems would look like once we follow the constructive approach. The stress on computation is however, only to the extent that the notion of error bound is incorporated in the definition of a number. But the question whether a program can be obtained to determine numbers within given error bound is not dealt with at all.

Despite its limitations, a positive contribution of the constructive approach is in bringing about the recognition that the phrase 'it exists' better be interpreted as'it can be computed'.

3.4.2 Interval analysis, as the name also suggests, takes intervals of real numbers as new kinds of numbers. In it a number is replaced as the basically by a pair of oral numbers [a, b] . The numbers a' and b' are refered to as left and right

end points. This indicates that computable number is also a parallel concept to interval analysis, where

$$[\, a,\ b\,] = \left[\, X(s) - \frac{1}{s},\ X(s) + \frac{1}{s}\,\right]$$

for the computable number X(s) representing a real number 'X' that has been represented as a,b in interval analysis.

In the literature [6] we find two reasons of interval arithmatic :

1. Exact Interval Arithmatics In which, we define operations on interval numbers as follows :

$$[\, a,\ b\,] + [\, c,\ d\,] = [a + c,\ b + d]$$

$$[\, a,\ b\,] - [\, c,\ d\,] = [a - c,\ b - d]$$

$$[\, a,\ b\,] \cdot [\, c,\ d\,] = [\min (ac,\ ad,\ bc,\ bd),\ \max (ac,\ ad,\ bc,\ bd)]$$

and $[a,\ b\,] / [\, c,\ d] = [a,\ b] \cdot \left[\, \frac{1}{d},\ \frac{1}{c}\right]$

2. Rounded Interval Arithmatics : This includes a practical way to find error bound, by taking differences between single precision and double precision results in interval analysis.

An important fact about interval analysis is that the Distibutive law does not hold for worth operations.

Let us see this with this example :

$$[1, 2]\ ([1, 2] - [1, 2]) \neq [1, 2][1, 2] - [1, 2][1, 2]$$

R.H.S. $= [1, 2]\ ([1, 2] - [1, 2])$

$= [1, 2]\ [-1, 1]$

$= [-2, 2]$

L.H.S. $= [1, 2][1, 2] - [1, 2][1, 2]$

$= [1, 4] - [1, 4]$

$= [-3, 3]$

Thus we have relation

$$[1, 2]\ ([1, 2] - [1, 2]) \subset [1, 2][1, 2] - [1,2][1, 2]$$

3.4.3 Computable Analysis

Aberth's book 'Computable Analysis' is the collection of work in this direction and this book is the main source for this thesis also. In this book a computable number is defined as a finite step program which with finite computation can produce the needed result with the required error-bound on it. Let us say we want to compute

$$y = f(x)$$

Then computable analysis will give as an input, the error bound on y and the computer program has to decide about the error bound on X(s). With this changed concept of error, computable analysis incorporates the constructive idea as it uses a program as a primitive element and computability on a finite state machine as a proof for any computable number.

3.4.4 Interval Analysis versus Computable Analysis : After this historical review, we can think about the relation between interval analysis and computable analysis as interval analysis has been tried out on computer [*] rather extensively. It would seem the algorithms and software techniques developed for interval analysis can be usefully employed in computable analysis.

In interval analysis, every number is represented by an interval say a, b for X such that $a < x < b$. For calculation of a function, independent variables are given as intervals. Calculation will take care of these intervals and resulting interval number is computed. This resultant interval is not known in advance.

Computable analysis on the other hand specifies the required maximum error bound on the result. In figure 3.1(a) we have seen that computable analysis have interval analysis as

part, so the work done in interval analysis area can help computable analysis. One such attempt will be made now.

3.4.5 Software Considerations : J.M. Yake in his paper 'Portable software for interval arithmatic' [7] , has indicated that three different versions of Interval Arithmatic Software are running on computers. We will enumerate some features which can help in developing the software for computable Arithmatic

(1) Brent's multiple precision package : In this package an initialization routine sets parameters as base precision and exponent range. Then it allows a programmer to use multiple precision but intermediate precision should be less than maximum precision fixed while initialization. This is apparently to carry out a chain of computations on a variable precision than the present fixed mode. This facility can be used in computable number programs as error-bound of inputs is varied till we get the required error bound on the result.

(2) Single precision Interval Arithmatic package : This package was interfaced with Brents multiple precision package.

This idea also can help in preparing computable analysis software in modular form. First module can give programs for standard functions as per need of the computable analysis.

This module can be interfaced with some variable precision module, analogous to the Brent's multiple precision package.

(3) Interval software with Directed rounding : Interval software with Directed rounding for interval arithmatic was implemented by writing software to perform floating point operations in machine language (preferably). Although not much detail is available about Directed rounding is provided but it seems to be meant to provide a facility to change rounding errors in the program itself.

(4) Nonstandard Variable : About precompiler they have indicated that Nonstandard variable were declared to specify interval number and precompiler is to generate standard Fortran programmes. In these operations and functions on nonstandard data elements are replaced by CALL's to needed modules of supporting packages.

This idea is also useful for the computable software as nonstandard type elements here will be 'computable numbers', in place of interval numbers.

More specific comments cannot be made at present, but in future these softwares of Interval Arithmatic can be obtained and examined for the above mentioned facilities. Computable software can be prepared on analogous lines which will make the use of computable analysis practically feasible.

Section 4

The Computable Number Field

4.1 Introduction

From the discussions of Sections2 and 3, it should be sufficiently clear why real numbers need to be replaced by computable numbers. We shall now, in this section examine the mathematical details about computable numbers. We shall in particular examine the fact that like the real numbers, computable numbers also constitute a field. We shall see how the various arithematic operations and relations for computable numbers are to be interpreted. In the last, concepts of computable functions, computable sequences and complex computable number will be discussed.

4.2 The Basic Concept

While talking about computable numbers in Section 1 we had broadly explained it to be a programme X(s) that produces from the value of s given as input, the value of the number **X** to within an error of $\pm 1/s$.

Now, what is the relationship between $X(s_1)$ and $X(s_2)$, the programme outputs for two different inputs s_1 and s_2?

To explain this, let us take a programme $\alpha(s)$ that computes the value of π. Let us say X(50) is to be computed, which means that the value of π is to be computed with maximum error $\pm$ 1/50. This is shown in figure

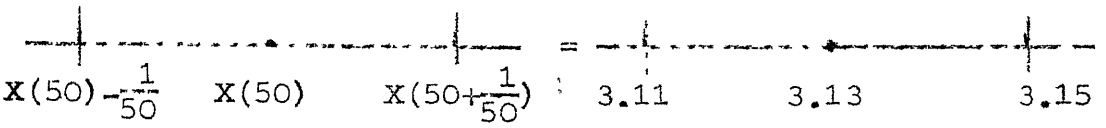


Fig. 4.1(a)

Likewise to compute X(100), the result should have a maximum error of $\pm$ 1/100. This can be represented graphically as in fig.

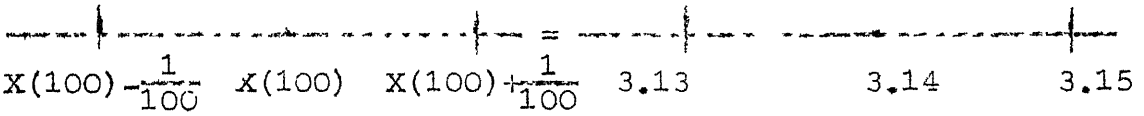


Fig. 4.1(b)

Together, X(100) and X(50) be can be represented as shown in Figure

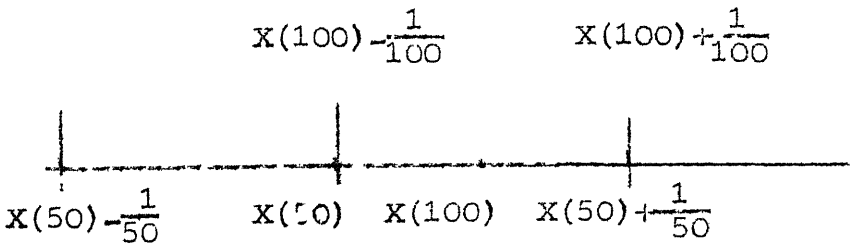


Fig.4.1(c)

Here we see that X(100) and X(50) have an overlapping region(dashed). Let us see what are the possible ways in

which the overlapping can take place. There are essentially three different ways as shown in figure 4.2.

For the number Y = 3, and its computable values Y(50) and Y(100) may be mutually disposed.

Case 1

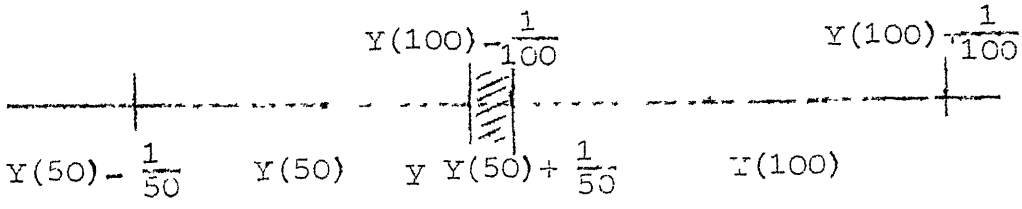


Fig. 4.2(a)

Case 2

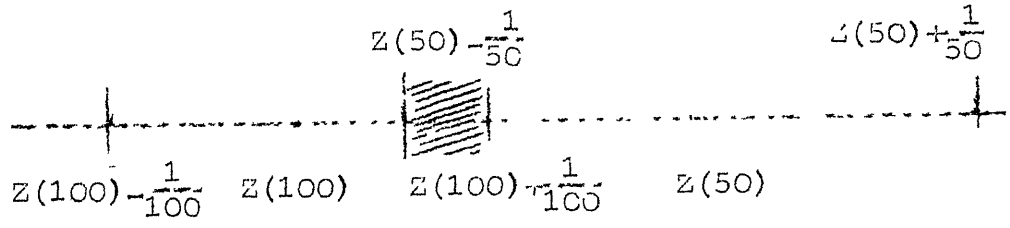


Fig. 4.2(b)

Case 3

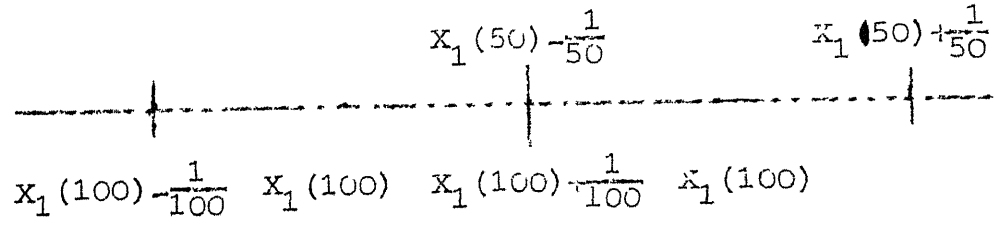


Fig. 4.2(c)

It can be seen that it is not possible for $X_2(50)$ and $X_2(100)$ to be such that

$$X_2(50) - X_2(100) \leq \frac{1}{100} + \frac{1}{50}$$

For, it were so then

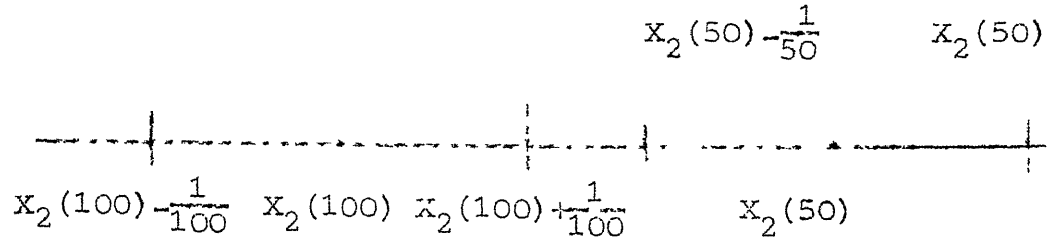


Fig.4.2(d)

But as shown in figure 4.2(d) the intervals of $X_2(100)$ and $X_2(50)$ do not overlap which is not possible if $X_2(100)$ and $X_2(50)$ are both to approximate the same number X.

With these clarifications, we are now ready to look into the precise definition of a computable number. As we presently see one first needs to introduce the notion of programmable function followed by that of a computable process and the equivalence of computable processes. Our presentation is based entirely on Aberth [1] .

Let P denote a programme on a URM and let $P(x_1,...x_n)$ denote the output of P when P halts. If P does not halt then $P(x_1,...x_n)$ is said to be undefined), where $x_1...x_n$ are values of the programme input variables at the starting point.

Let $\varphi(x_1,..x_n)$ be a rational valued function of the rational variable $x_1,..x_n$ and let P be the corresponding programme such that $P(x_1,..x_n)$ is defined if and only if

$\varphi(x_1,..x_n)$ is defined, we say that φ is a programmable function if

$$P(x_1,...x_n)=\varphi(x_1,...x_n)$$

The programme P is said to realize φ.

A computable process is a programmable function $\alpha(s)$, that is defined for all postive integers s and is such that for any two positive integers s and t

$$|\alpha(s) - \alpha(t)| \leq \frac{1}{s} + \frac{1}{t} \qquad .. (4.1)$$

we may note at this point that the justification of the inequality (4.1) lies in the arguments of the overlapping regions given at the beginning of this discussion.

Let α and ω' be two computable processes, we say, that ω and ω' are equivalent, $\omega \equiv \omega'$ if for any positive integers s and t

$$|\omega(s) - \omega'(t)| \leq \frac{1}{s} + \frac{1}{t} \qquad (4.2)$$

Condition 4.2 states the equivalence relation of computable processes α and ω', which is reflexive transitive and commutative in nature.

We may now finally define a computable number as follows.

The equivalence class of a computable process is a computable number.

For the sake of notation convenience we shall use X(s) to define a computable number approximating X.

4.3 Error Representation in Computable Number

So far we have interpreted X(s) to mean the value of real number X to within an error $\pm$ 1/s. There are other equally useful ways of describing error in X(s). For instance X(s) is a computable number then we may choose to define error equal to $\pm \log_{10} s$. Likewise on, instead of specifying s, may specify E = 1/s. All these choices are matter of notation convenience only.

4.4 The inequality Relation

To be able to fully define the arithematic of computable number it is now essential to formulate various arithematic operations we first take up the inequality relation.

Let us see some examples to understand the notion of inequality in computable numbers.

Let $X_2(s)$ be a programme to compute $\sqrt{2}$ as given below :

1. Value of s given
2. n = 1
3. $(n/s)^2 - 2$ if positive, n/s is the result otherwise go to step 4.
4. n = n+1
5. Go to Step 3

Let us say $Y_2(s)$ is the computable number approximating $\sqrt{2}$ produced by the algorithm given below

1. Given value of s_1
2. x = o, r = o
3. n = 1
4. $(X+n.10^r)-2.0$ if negative n = n+1, GO TO 3

   otherwise $x = x+(n-1)\ 10^r$

   $x = x - 1$

   GO TO 3 till $x = -s_1$

(Here $x = 10^{s_1}$).

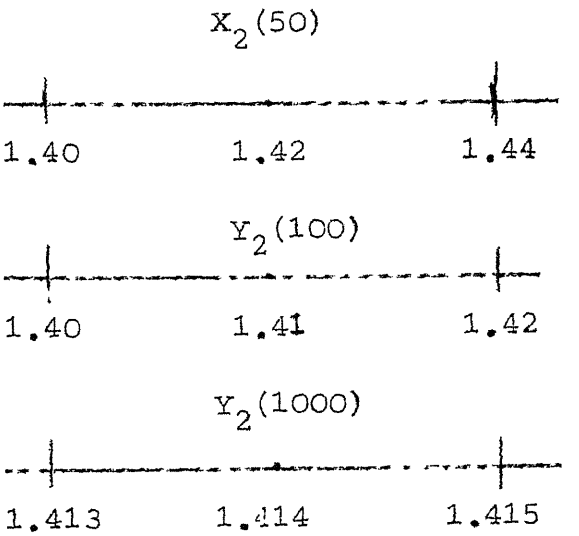
Now let us search if $X_2(s)$ is greater than, equal to or less than $Y_2(t)$ for 's' and 't' positive integers, with these results

$X_2(50) = 1.42$

$Y_2(100) = 1.41$

$X_2(50) = 1.42$

$Y_2(1000) = 1.414$



As we see from the diagrams, since $X_2(s)$ and $Y_2(t)$ must contain the number X, that two will always overlap. When for any arbitrary values s and t, if such overlap exists for $X_2(s)$ and $X_2(t)$ then we say they are equal otherwise they are unequal.

If there is a choice of some s and t such that overlap does not take place, then we say two computable numbers unequal. In particular if $X_2(s)$ and $X_3(t)$ are two computable numbers then we say $X_2(s) < X_3(t)$ if for some choice of s and t, $X_2(s) + \frac{1}{s} < X_3(t) - \frac{1}{t}$. We can alternatively say that $X_3(t) > X_2(s)$.

4.5 Operation on Computable Numbers

Let us now turn to arithematic operations on computable numbers. First we need to define arithematic operation on computable processes.

Definition : If α and β are computable processes and operation 'O' represents one of the rational operations $+$, $-$, $\div$ and $*$, then $\alpha O \beta$ is the process given by

$$(\alpha O \beta)(s) = \alpha(s_1) \; O \; \beta(s_1)$$

where

$$s_1 = \min_{v \geq 1} v \quad \{\max \left| \alpha(v) O \beta(v) - \alpha(v) \pm \frac{1}{v}) O (\beta(v) \pm \frac{1}{v}) \right| \leq \frac{1}{s} \}$$

It can be shown that $\alpha O \beta$ is also a computable process. Here division operation holds only when $\beta(s_1) \neq 0$. Thus operations between two computable numbers are related with error computation and control methods also.

Definition : Let α denote the computable i.e. the equivalent class of a computable process α. Let 'O' denote one of the operations $+, -, *$ and then $\alpha \; \beta$ is the computable process $\alpha O \beta$. For divide operation, to hold we must have $\beta(s_1) \neq 0$ for any positive integer s_1.

On the same lines, we can reformulate the concepts for computable functions and computable sequences.

4.6 Comoutable function and sequence

Considering the computable number as a programme the concept of function has to be recast in cerms of programmes and computable processes.

Consider as an illustration the following function in real analysis.

$$F(x) = x + x^2 + \sqrt{3}$$

Here x is represented as S(x). the computable numbers approximating X. Further let Y(s) be the program to compute $\sqrt{3}$ with $\pm 1/s$ error bound.

Then the computable function corresponding to F(x)

$$F(X(s_1),s) = X(s_1) + X_1^2(s_1) + Y(s_1)$$

where

$$\frac{1}{s} = \max\left| X(s_1)+X^2(s_1)+Y(s_1)\pm\frac{1}{s_1}\right)-\left(X(s_1)\pm\frac{1}{s_1}\right)^2-Y(s_1)\pm\frac{1}{s_1}\Big)\Big|$$

$$\text{or } \frac{1}{s} = \frac{2}{s_1} + \frac{2\,X(s_1)}{s_1} + \frac{1}{s_1^2}$$

$$\text{Here } s_1 = \min_{v\ 1} v\left\{\max\left|F(x(v)) - F\left(x(v)\pm\frac{1}{v}\right)\right| \le \frac{1}{s}\right\}$$

Thus error control method is just parallel to used for the operations on computable numbers. Sequence when used with computable numbers will have changed formulation. This we will see now.

In engineering or scientific work sequences occur very commonly. A precise definition of computable sequence is given as follows.

Let $\omega_n(s)$ be a programmable function, when n is a positive integer, such that for a fixed n, $\alpha_n(s)$ is a computable process. We can say that $\omega_n(s)$ defines a sequence a_n where $a_n = \omega_n$.

It can be shown that if a_n is a sequence that if a_n is a sequence then so are the b_n and c_n given by

$$b_n = \sum_{i=1}^{n} a_i \quad \text{and} \quad c_n = \prod_{i=1}^{m} a_i$$

This result is of basic important in the analysis of digital filter algorithms.

4.7 Complex Computable Analysis

Once we replace real numbers by computable numbers we can extend the notion of 'computable' in complex field also

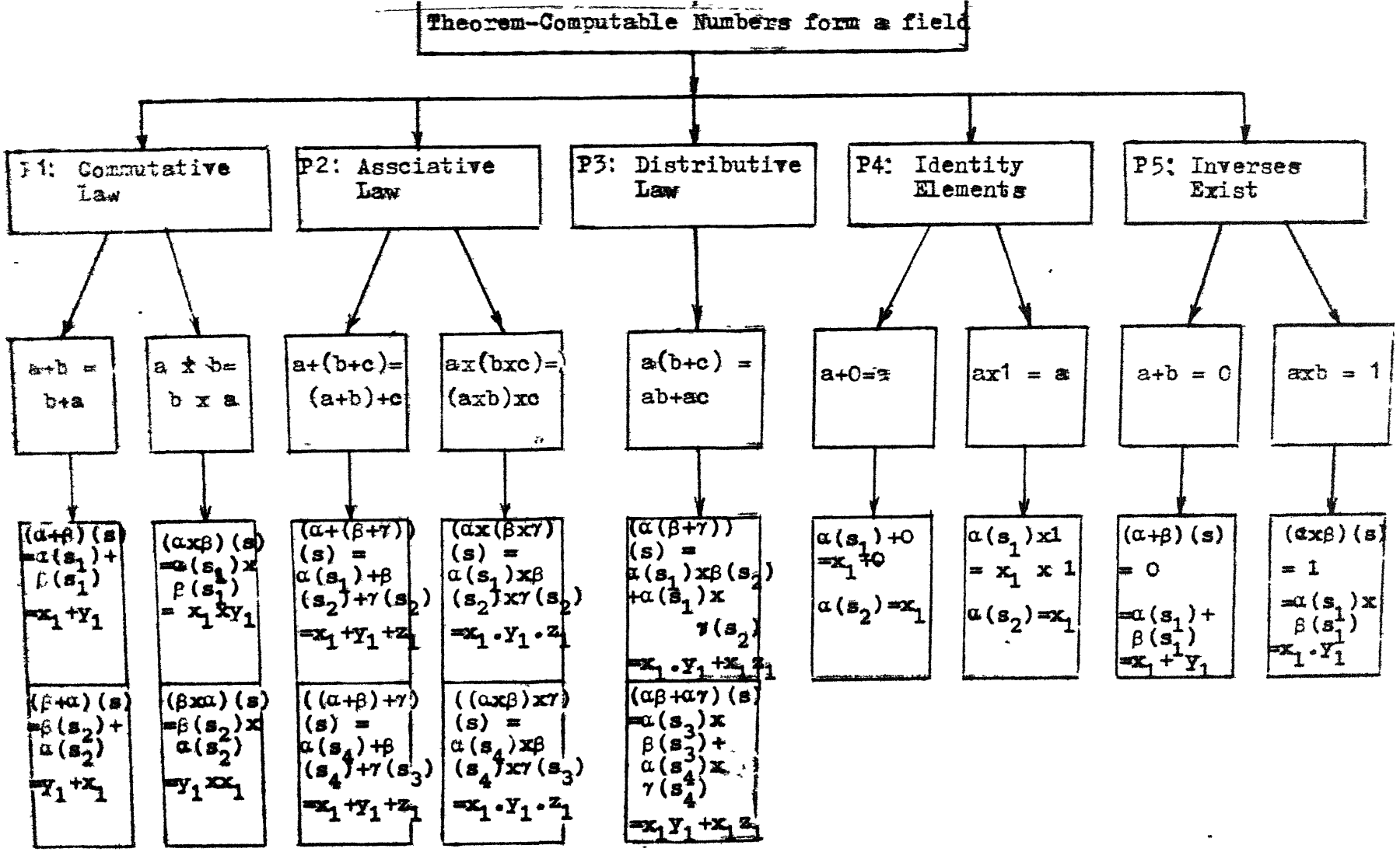
complex numbers can be replaced by 'computable complex numbers'. And then parallel to computable analysis, a computer computable analysis can be worked out.

A complex number is given by a pair of real numbers (a_1, a_2), where a_1 is called real part and a_2 - as imaginary part. A computable complex number is given by a pair of computable numbers (A_1, A_2) as in a complex number, here A_1 is a programme $\omega_1(s_1)$, as real part and A_2 - another programme $\omega_2(s_1)$ as imaginary part.

Now we come to most crusial result namely that the computable numbers together with operations defined above form a field. This is known as Rice Theorem [1] . In view of its importance, we present here a structural version of the proof of this theorem.

4.8 Structured Proof of Rice-Theorem

The structuring of the proof is based on the guide lines given by Leren [9] to make it more readable. The basic idea of structuring is as follows, proof is broked into different levels and each level will be able to convince the reader, what next is needed to prove the theorem. We will see



Note - In Fourth State, we take x_1 - From overlap region of $\alpha(s_1)$, $\alpha(s_2)$, $\alpha(s_4)$

y_1 - From overlap region of $\beta(s_1)$, $\beta(s_2)$, $\beta(s_3)$ and $\beta(s_4)$

z_1 - From overlap region of $\gamma(s_1)$, $\gamma(s_2)$, $\gamma(s_3)$ and $\gamma(s_4)$

Fig. 4.3 Structure of Rice Theorems Proof

this with Rice-theorem's proof

Level 1 - Theorem is computable numbers form a field.

Level 2 - In level 1, to prove that computable numbers form a field, we should try to prove that computable numbers fullfil all conditions necessary to call it a field, these conditions are that computable numbers obey

P1 Commutative Law

P2 Associative Law

P3 Distributive Law

P4 Identity Law and

P5 Inverses exist

So now in next levels we need to prove that above conditions are satisfied for the computable numbers.

Level 3 : To prove it to P5 propositions or conditions we will try to see for each proposition, how to prove it and what things will be need to prove it.

For (P1) : To prove that computable number obey commutative law we have to prove that computable numbers obey these relations:

P1.1 $a+b = b + a$

P1.2 $a \times b = b \times a$

where a, & b both are computable numbers.

For (P2) : If associative law holds good for the computable number or not, can be known if we can prove that computable numbers obey these two relations:

P2.1 $a+(b+c) = (a+b)+c$

P2.2 $a \times (b \times c) = (a \times b) \times c$

where a, b, and c are computable numbers.

For P3 : For computable numbers, if one can prove that for two computable number a and b

$$a(b+c) = ab + a.c.$$

holds then distributive property for computable numbers is satisfied.

From P4 : For proving that computable numbers have identity elements, relations

P4.1 $A+0 = a$

P4.2 $bx1 = b$

should hold where a and b are computable numbers.

Fro P5 : For the last property one has to prove that computable numbers have their inverses, in other words relations

$$a + b = 0$$

$$a \times b = 1$$

for a and b computable numbers.

Level 4 : In level 3 we came to know, what is needed to prove the proposition of level 2. Now we will see each proposition and will prove conditions required (by level 3) for computable numbers.

For P1 : From level 3 relations to be proved are

P1.1 $a + b = b + a$

P1.2 $a \times b = b \times a$

where a and b both are computable numbers.

P1.1

Relation is

$$a + b = b + a.$$

As a and b are computable numbers, let us represent these by $\alpha(s)$ and $\beta(s)$ computable programmes, so

$$(\alpha + \beta)(s) = \alpha(s_1) + \beta(s_1) \qquad (4.1)$$

$$(\beta + \alpha)(s) = \beta(s_2) + \alpha(s_2) \qquad (4.2)$$

To prove $(\alpha + \beta)(s) = (\beta + \alpha)(s)$

we need to prove

$$\alpha(s_1) + \beta(s_1) = \beta(s_2) + \alpha(s_2)$$

We choose Y value such that it is in overlap region of $\alpha(s_1)$ and $\alpha(s_2)$ and 'Z' value from overlap region of $\beta(s_1)$

and $\beta(s_2)$ and putting these values in (4.1) and (4.2) equations.

$$\alpha(s_1) + \beta(s_1) = Y + Z$$

$$\beta(s_2) + \alpha(s_1) = Z + Y$$

Since $Y + Z = Z + Y$ for normal plus operation.

P1.2 : On same lines

$$(\alpha * \beta)(s) = \alpha(s_2) * \beta(s_3)$$

$$(\beta * \alpha)(s) = B(s_4) * \alpha(s_4)$$

Let M and N be two values from overlap region of $\alpha(s_3)$, $\alpha(s_4)$ and $\beta(s_3$, $\beta(s_4)$ respectively

$$\therefore \alpha(s_3) * \beta(s_3) = M * N = N * M = \beta(s_4) * \alpha(s_4)$$

For P2 : To prove Associative law for computable numbers rea relations to be proved are

P2.1 $a + (b + c) = (a + b) + c$

P2.2 $a \times (b \times c) = (a \times b) \times c$

where a,b,c are computable number, which can put as $\alpha(s)$, $\beta(s)$ and $r(s)$ respectively.

P2.1 Relation is

$$a + (b + c) = (a + b) + c$$

or $(\alpha + (\beta + r))(s) = ((\alpha + \beta) + r)(s)$

Using $(\alpha O\beta)(s) = \alpha(u_s) O \beta(u_s)$ relation

RHS $(\alpha+(\beta+r))(s) = \alpha(s_1)+(\beta+r)(s_1)$

$= \alpha(s_1) + \beta(s_2) + r(s_2)$ (4.3)

LHS $((\alpha+\beta) + r)(s) = (\alpha+\beta)(s_3+r(s_3)$ (4.4)

$= \alpha(s_4) + \beta(s_4) + r(s_3)$

Taking P value from overlap region of $\alpha(s_4)$ and $\alpha(s_1)$,

Q value from overlap region of $\beta(s_4)$ and $\beta(s_2)$

and R value from overlap region of $r(s_3)$ and $r(s_2)$

From equations (4.3) and (4.4)

$$\alpha(s_1) + \beta(s_2) + r(s_2) = P + Q + R$$

$$\alpha(s_4) + \beta(s_4) + r(s_3) = P + Q + R$$

Hence $(\alpha+(\beta+r))(s) = ((\alpha+\beta)+r)(s)$

P2.2 : $ax(bxc) = (axb)xc$

or $(\alpha*\beta*r))(s) = ((\alpha*\beta)*r)(s)$

On the same lines .

$$(\alpha*(\beta*r))(s) = \alpha(s_1)*(\beta*r)(s_1)$$
$$= \alpha(s_1)*\beta(s_2)*r(s_2) \qquad (4.5)$$

$$((\alpha*\beta)*r(s) = (\alpha*\beta)(s_3)*r(s_3)$$
$$= \alpha(s_4)*\beta(s_4)*r(s_3) \qquad (4.6)$$

Selecting P,Q,R as above mentioned -

From equations 4.5 and 4.6

$$\alpha(s_1)*\beta(s_2)*r(s_2) = P*Q*R$$

$$\alpha(s_4)*\beta(s_4)*r(s_3) = P*Q*R$$

Hence $(\alpha*(\beta+r))(s) = ((\alpha*\beta)*r)(s)$

For P3 : To prove Distributive Law for computable numbers; relation to be proved are :

P3.1 $\quad a(b+c) = a.b + a.c$

where a, b and c are computable numbers representable $\alpha(s)$, $\beta(s)$, respectively.

Relation to be proved is

$$a(b+c) = a.b + a.c$$

or $$(\alpha.(\beta+r))(s) = (\alpha.\beta + \alpha.r)(s)$$

R.H.S.

$$\begin{aligned}(\alpha*(\beta+r))(s) &= \alpha(s_1) * (\beta+r)(s_1) \\ &= \alpha(s_1)*(\beta(s_2)+r(s_2)) \\ &= \alpha(s_1)*\beta(s_2) + \alpha(s_1)*r(s_2) \end{aligned} \tag{4.7}$$

L.H.S.

$$\begin{aligned}(\alpha*\beta+\alpha*r)(s) &= (\alpha*\beta)(s_3)+(\alpha*r)(s_3) \\ &= \alpha(s_4)*\beta(s_4)+\alpha(s_5)*r(s_5) \end{aligned} \tag{4.8}$$

Selecting P value from overlap region of $\alpha(s_1)\alpha(s_4)$ and $\alpha(s_5)$

Q value from overlap region of $\beta(s_2)$ and $\beta(s_4)$

and R value from overlap region of $r(s_2)$ and $r(s_5)$

From equation (4.7) and (4.8)

$$\alpha(s_1)*\beta(s_2)+\alpha(s_1)*r(s_2) = P*Q+P*R$$
$$\alpha(s_1)*\beta(s_4)+\alpha(s_5)*r(s_5) = P*Q+P*R$$

Hence $(\alpha*(\beta+r))(s) = (\alpha*\beta+\alpha*r)(s)$

For P4 : To see if identity elements exist for the computable number set, relation to be proved are

P4.1 - $a + 0 = a$

P4.2 - $a \times 1 = a$

where 'a' is a computable, which we can represent by $\alpha(s)$:,

Here $(\alpha + 0) = \alpha(s_1) + 0$
$= \alpha(s_1)$

and $(\alpha*1)(s) = \alpha(s_2)\ 1$
$= \alpha(s_2)$

For P5 : Property of 'esistence of inverses' can be proved by the relations

$a + b = 0$

$a \times b = 1$

where a and b are two computable numbers and we represent them as $\alpha(s)$ and $\beta(s)$ respectively.

P5.1 :

$$a + b = 0$$

$$\text{or} \quad (\alpha+\beta)(s) = 0$$

$$\text{or} \quad \alpha(s_1) + \beta(s_1) = 0 \qquad (4.9)$$

P5.2 :

$$a \times b = 1$$

$$\text{or} \quad (\alpha*\beta)(s) = 1$$

$$\text{or} \quad \alpha(s_2)*\beta(s_2) = 1 \qquad (4.10)$$

Seeing results (4.9) and (4.10), we can say that computable number set fulfils property of existence of inverses.

Section 5

How To Make It Practical

5.1 Introduction

To be able to make use of computable analysis for computational purposes, we have to look for practical approaches also. But it must not increase the complexity of programme for solving numerical problems at the same time computational algorithms based on it should automatically provide an indication of the error bound at each stage.

Some of the modifications needed for the purposes of general computable number programs would be the following ones :

1. Library functions will have to be rewritten.
2. operations +, -, * and ÷ will have to be programmed in computable number sense.
3. Programmes for the comparision of two computable numbers as well as for binary relations in general, will have to be reformulated.
4. Infinite series expansions as trigonometric functions will have to be constructively redefined.

5. In working backwards from the accuracy of the result of an algorithm operation, an appropriate rule will have to be evolved for distributing it to operands.
6. Modularity in programmes will become all the more important.
7. In view of the increased computational work that is required to incorporate error as a parameter, method of cutting down the number of iterations required for computing a function to within a given error bound will have to be devised.

Let us now discuss the need for each modification and some possible ways to carryout that modification.

5.2.1 Library functions

Library functions on computers allow us to compute standard functions with most accurate possible results at present. For computable analysis we have to rewrite these library functions such that the programme should be able to compute the result for the given final error bound.

For this, solution lies in arranging for the computation of error bound and error controlling feedback. Let us illustrate this with the help of an example. Let us compute

$e^{0.5}$ with $\pm \frac{1}{100}$ error bound.

Now $e^{0.5}$ can be put in its expanded form as follows

$$e^{0.5} = 1 + 0.5 + \frac{(0.5)^2}{\underline{2}} + \frac{(0.5)^3}{\underline{3}} + \ldots + \frac{(0.5)^n}{\underline{n}} + \ldots$$

For having $\pm \frac{1}{100}$ error bound, this series has to be truncated at the n^{th} term such that

$$\frac{1}{100} < \frac{(0.5)^{n+1}}{\underline{n+1}} + \frac{(0.5)^{n+2}}{\underline{n+2}} + \ldots$$

Thus algorithmic steps needed are as follows :

1. start summation from 1^{st} term
2. compute next term
3. Add it to the sum
4. Find error due to truncation on that term.
5. check if truncation error $\frac{1}{100}$
6. if not go to step 2

   else stop execution.

In this procedure truncation error has to be computed. One way to do this is as follows

$$E_n = \frac{(0.5)^{n+1}}{\underline{n+1}} \left(1 + \frac{0.5}{n+2} + \frac{(0.5)^2}{(n+2)(n+3)} + \ldots \right)$$

$$= \frac{(0.5)^{n+1}}{\underline{n+1}} (1 + r_1 + r_1 . r_2 + r_1 . r_2 . r_3 + \ldots)$$

where $r_1 = \frac{0.5}{N+2}$, $r_2 = \frac{0.5}{n+3}$, $r_3 = \frac{0.5}{n+4}$, ... etc.

As we have $r_1 > r_2 > r_3$ ... for n, a positive integer, by putting $r_1 = r_2 = r_3$ we can write

$$E_n \leq \frac{(0.t)^{n+1}}{\lfloor n+1} \quad (1 + r_1 + r_1^2 + r_1^3 + \ldots)$$

or $$E_n \leq \frac{(0.5)^{n+1}}{\lfloor n+1} \cdot \frac{1}{1-r_1}$$

(as $r_1 < 1$ for a converging series)

This $\frac{(0.5)^{n+1}}{\lfloor n+1} \cdot \frac{1}{1-r_1}$ value can be taken as the truncation error by truncation of series at n^{th} term.

Such algorithms have to be written for each library function. Once such library functions are available, computable number programmes can use them directly by calling these library functions when ever required.

5.2.2 Operations

In section 4, we pointed out that for the computable numbers an arithematic operation is of the form

$$Z(s) = X(s_1) \; O \; Y(s_1)$$

where $X(s_1)$, $Y(s_1)$ and $Z(s)$ are computable numbers and 'o' denotes an arithematic operation. Further, s_1 is related to s as

$$s_1 = \min_{v \geq 1} v \{ \max | X(v) o\ Y(v) - (X(v) \pm \frac{1}{v}) o\ (Y(v) \pm \frac{1}{v}) | \leq \frac{1}{s} \}$$

This means that an arithematic operation in computable analysis has to be treated differently and not as we carryout an arithematic operation on two MAP real numbers. To be more specific, we will have to include in the program implementing that operation a provision for obtaining from the value of s the error in the final, the error to be allowed in the operand of the intermediate steps. To give an example, consider the arithematic expression

$$Y = x + \frac{5x^2}{13} + - \frac{x}{3}$$

In terms of computable numbers, this becomes

$$Y(s) = x(s_1) + \frac{5 * x^2(s_2)}{13} + \frac{x(s_3)}{3}$$

Suppose it is given that s = 100 and $X(s_1)$ gives the value of real number Sin 40^o with $\pm \frac{1}{s_1}$ error. To compute Y(s) we can proceed as follows :

1. Here, there are two binary operations involving three terms. Let us distribute the error $\frac{1}{s}$ equally. This means that every term should be computed with $\pm \frac{1}{300}$ error bound.

2. Taking each term one by one.

   (a) $X(s_1)$ will be computed for $\frac{1}{s_1} = \frac{1}{300}$

   (b) $\frac{5}{13} * X^2(s_2)$ will be computed using computable number algorithm for the operation '*', choosing s_2 such that total error on $\frac{5}{13} * X^2(s_2)$ is $\frac{1}{300}$.

   (c) Swuare root $(X(s_3))^{1/2}$ is computed using library function programme 'square root' such that s_3 be selected to have the error bound $\frac{1}{300}$ on $\frac{X(s_3) \cdot 5}{3}$.

This way computable operations will have to be carried out using a separate algorithm which can control error to the required value.

5.2.3 Comparision of two computable numbers

For comparing two computable numbers, we have to call some subroutine to decide whether one computable number is greater than or equal to or less than the other computable number.

Let us, say X, Y are two real numbers represented by computable numbers X(s) and Y(s) respectively. We can have

three possible cases :

1. If $X(s) + \frac{1}{s} < Y(s) - \frac{1}{s}$, then $X(s) < Y(s)$, this can be graphically shown as in figure 5.1(a)

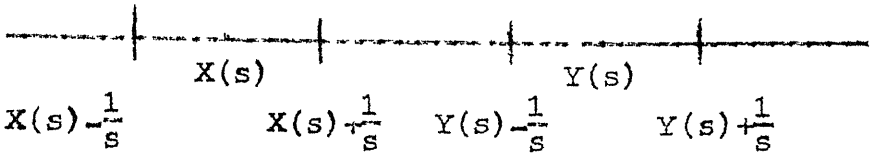


Fig. 5.1(a)

2. If $X(s) - \frac{1}{s} > Y(s) + \frac{1}{s}$ then $X(s) > Y(s)$, this can be graphically represented as in figure 5.1(b)

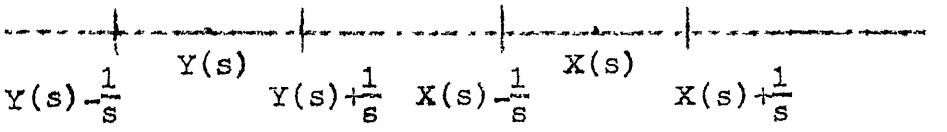


Fig. 5.1(b)

3. If $X(s) + \frac{1}{s} > Y(s) - \frac{1}{s}$ and $X(s) - \frac{1}{s} < Y(s) + \frac{1}{s}$ then $X(s) = Y(s)$

This is shown graphically represented as in figure 5.1(c)

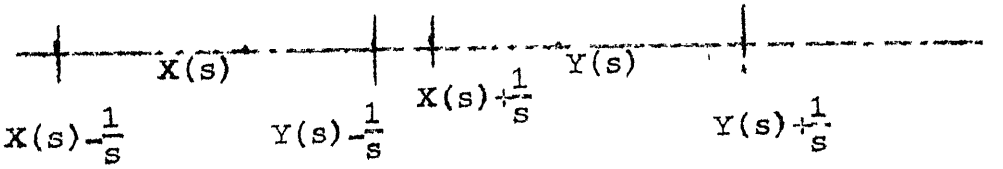


Fig. 5.1(c)

Thus all three inequality relations have to be tackled in the new sense.

5.2.4 Trigonometric functions

Trigonometric functions in general have infinite terms in the expanded form. We will discuss here a case of convergent series with infinite terms and see what the problem is in computing error bounds and we will try to see if there are reasonable solutions.

To understand the problem, let us consider an example to compute log(1+X(s), where a real number X is represented by X(s), a computable number. An expanded form can be as follows

$$\log (1 + X(s)) = X(s) - \frac{X^2(s)}{2} + \frac{X^3(s)}{3} \ldots$$

In such a case there are two variables in error computation. The first one is error bound $\pm \frac{1}{s_1}$, on $X(s_1)$ as per the computable number concept discussed in Section 1. Another one is the number of terms 'n' after which truncation is done. So the problem is to decide the trade off between these two errors such that the summation of both is equal to the error required. In this respect first of all we will see the contributions of each error separately for the function SinX as an example.

For the function SinX, when we represent the real number 'X' by a computable number X(s), the error due to truncation and the error contributed by error bound $\pm \frac{1}{s}$ on X(s) have to be computed we will examine the error from one source with the other error assumed to be zero.

Series expansion of SinX is as follows.

$$\text{SinX} = X - \frac{x^3}{\lfloor 3} + \frac{x^5}{\lfloor 5} - \frac{x^7}{\lfloor 7} + \ldots \frac{x^{2n-1}}{\lfloor 2n-1} + \ldots$$

Case 1. when $\frac{1}{s} = 0$ or $X(s) = X$

In the series expansion

$$T_n = \frac{x^{2n-1}}{\lfloor 2n-1}, \quad T_{n+1} = \frac{x^{2n+1}}{\lfloor 2n+1}$$

$$\therefore \frac{T_{n+1}}{T_n} = - \frac{x^2}{2n(2n+1)} = r_1$$

For summation of first n terms.

$$\text{SINX} = X - \frac{x^3}{\lfloor 3} + \frac{x^5}{\lfloor 5} - \ldots + \frac{x^{2n-1}}{\lfloor 2n-1} - \frac{x^{2n+1}}{\lfloor 2n+1} + \ldots$$

$$= \text{SM} + \frac{x^{2n+1}}{\lfloor 2n+1} \left(1 - \frac{x^2}{(2n+2)(2n+1)} + \frac{x^4}{(2n+2)(2n+3)(2n+4)(2n+5)} + \ldots \right)$$

$$= \text{SM} + \frac{x^{2n+1}}{\lfloor 2n+1} \left(1 + r_1 + r_1^2 + r_1^3 + \ldots \right)$$

where $\frac{T_{n+1}}{T_n} = r_1$, $\frac{T_{n+2}}{T_n} = r_1^2$ etc

since $\frac{T_{n+1}}{T_n} > \frac{T_{n+2}}{T_{n+1}} > \frac{T_{n+3}}{T_{n+2}}$ etc so upper bound of error

$$E_T = \frac{X^{2n+1}}{\lfloor 2n+1} (1 + r_1 + r_1^2 + r_1^3 + \dots) = \frac{X^{2n+1}}{\lfloor 2n+1} \cdot \frac{1}{1-r_1} \quad \dots\dots \quad (5.1)$$

Here $r_1 < 1$ for series is convergent one.

Case 2.

When the real number 'X' is represented by X(s) with a finite error bound $\pm \frac{1}{s}$. For this truncation error is taken to be zero, to see the effect of error bound $\pm \frac{1}{s}$, alone. We have here the series expansion

$$\text{Sin}X\ (s) = X(s) - \frac{X^3(s)}{\lfloor 3} + \frac{X^5(s)}{\lfloor 5} \quad \dots$$

$$\text{Sin}\ (X(s) \pm \tfrac{1}{s}) = X(s) \pm \frac{1}{s} - \frac{(X(s) \pm \frac{1}{s})^3}{\lfloor 3} + \frac{(X(s) \pm \frac{1}{s})^5}{\lfloor 5} \quad \dots$$

Thus

$$E_s = \max \text{Sin}\ (X(s) \pm \tfrac{1}{s}) - \text{Sin}\ X(s)$$

$$= \frac{1}{s} - \frac{3X^2(s)}{2.s} - \frac{3X(s)}{2s^2} + \frac{1}{6s^3} + \frac{5X^4(s)}{120.s} + \frac{10X^3(s)}{120.s^2} + \dots \quad .. \quad (5.2)$$

E_T represents the truncation error and E_s represents the error due to $\frac{1}{s}$ error in X(s). The total error E_R can be

taken, as a first approximation, to be

$$E_R = E_T + E_s$$

Equations 5.1 and 5.2 show that as the number of terms, n, is increased whereas E_s increases. On the otherhand by increasing 's' in $X(s)$, one can reduce E_s and so that a higher E_T by summing fewer terms of the series may be This makes us think of many possible solutions for E_T and E_s, such that E_r is constant. So the question of selection of one pair of E_T and E_s out of many has to be solved for minimum.

Algorithms to compute infinite series that take into account this tradeoff between E_T and $E_s$computable number programmes have to be devised.

5.2.5 Equal distribution of error

In operations with computable numbers of a distribution of error bounds to constituent operands is necessary.

$$Z(s) = X(s_1) \; O \; Y(s_2)$$

where 'O' represents any of the operation +, -, * or $\div$ on computable numbers $X(s_1)$ and $Y(s_2)$. Here the error $\frac{1}{s}$ on $Z(s)$ will be decided by $\frac{1}{s_1}$ and $\frac{1}{s_2}$ the error bounds on $X(s_1)$ and

$Y(s_2)$ respectively. Several different pairs of s_1 and s_2 will produce the same value of s. Thus a well defined method has to be prescribed to select one of many choices. We will not give an example to help evolve a method to select an appropriate s_1, s_2 pair.

Let us compute $Y(100) = \sqrt{2} + \sqrt{3}$. This is particular of the form

$Z(s) = X_2(s_1) + X_3(s_2)$. We will take

$\sqrt{2} = 1.414 \ldots$, $\sqrt{3} = 1.732 \ldots$ and $\sqrt{2} + \sqrt{3} = 3.156 \ldots$

So we write some computable numbers as follows

$X_2(50) = 1.3$	$X_2(100) = 1.41$	$X_2(200) = 1.415$
$X_2(500) = 1.412$,	$X_2(1000) = 1.414$	
$X_3(50) = 1.6$	$X_3(100) = 1.73$	$X_3(200) = 1.735$
$X_3(500) = 1.734$	$X_3(1000) = 1.732$	

A few cases are now computed as follows

(1) $X_2(100) + X_3(500) = 1.41 + 1.734 = 3.144$ (s=83.3)

(2) $X_2(200) + X_3(200) = 1.415 + 1.735 = 3.150$ (s=100)

(3) $X_2(500) + X_3(100) = 1.412 + 1.73 = 3.142$ (s=83.3)

(4) $X_2(50) + X_3(1000) = 1.3 + 1.732 = 3.032$ (s=47.6)

(5) $X_2(1000) + X_3(50) = 1.414 + 1.6 = 3.014$ (s=47.6)

So these cases show that lows, requires high s_2 for the same s and conversely a low s_2 requires a high s_1. A appropriate selection can be $s_1 = s_2 = 200$

In general, for an addition of the form

$$Z(s) = X_2(s_1) + X_3(s_2) \tag{5.3}$$

in which the errors are related as

$$\frac{1}{s} = \frac{1}{s_1} + \frac{1}{s_2}$$

it is appropriate to choose $s_1 = s_2 = \frac{s}{2}$. This choice is based on the argument that the error distribute should bring down the computations for each of the operands. For any number $X(s)$, the smaller eh error $\frac{1}{s}$, the more the computations required. What we want that for both the operationals $X_2(s_1)$ and $X_3(s_2)$ the s_1 and s_2 be so shared that while $\frac{1}{s} = \frac{1}{s_1} + \frac{1}{s_2}$ is constant, $s_1 + s_2$ be minimised so that the computation for $X_2(s_1)$ and $X_3(s_2)$ are minimised. This implies that $s_1 = s_2$.

Consider now the multiplication in place of addition in eqn. 5.3 Assuming $X_2(s_1) \simeq X_3(s_2)$ and s_1, s_2 large integers,

$$\frac{1}{s} \simeq \left(\frac{1}{s_1} + \frac{1}{s_2}\right) X_2(s_1) + \frac{1}{s_1 s_2}$$

$$\simeq \left(\frac{1}{s_1} + \frac{1}{s_2}\right) X_2(s_1)$$

Thus here too $s_1 = s_2$ will require the least computational work.

Here one question may arise, if this convention will held good in general the answer is that except in some very exceptional cases, it will always had good. Two exceptional cases are enumerated here.

1. when $X_2(s_1)$ $X_3(s_2)$ then s will be controlled by the larger value.
   $$\frac{1}{s} = \frac{X_3(s_2)}{s_1} + \frac{1}{s_1 s_2}$$

2. Certain cases where $X_2(s)$ is a shorter programme than $X_3(s)$ or in other word $X_3(s)$ will require more computational work than that for $X_2(s)$.

In such cases, one can choose the distribution of errors such that

$$\frac{X_2(s_1)}{s_1} \leq \frac{X(s_2)}{s_2}$$

5.2.6 Modularity of Computable number Prgramme

Modularity of a programme helps in writing and also in understanding programmes. How this feature is useful in

the computable number programmes will be examined with help of three different cases.

1. Let us compute $\sqrt{2}$. This can be represents as a computable number $X_2(s)$. Here the computable number programme will use '2' and 's' as the input variables and will give a value after execution with $\pm \frac{1}{s}$ error bound.

2. Let us compute $\sqrt{2} \sqrt{x} 3$ which can be represented as

$$Z(x) = X_2(s_1) * X_3(s_1)$$

Here $X_2(s_1)$ and $X_s(s_1)$ are two computable numbers of the type mentioned in case (1). These are used to compute a new computable number $Z(s)$.

3. Let us say we want to compute

$Y = \sin (\sqrt{2} * (1 + \sqrt{3}))$, which can be represented in computable analysis as follows

$$y(s) = \sin (X_2(s_1) * (1 + X_3(s_1))$$

Here $X_2(s_1) * (1 + X_3(s_1))$ is a computable number of type mention in case (2). That uses $X_2(s_1)$ computable numbers of type given in case (1).

If $\quad Z_1(s_2) = X_2(s_1) * (1 + X_3(s_1))$

then $\quad Y(s) = \sin X(s_2)$

This Way for given 's' s_2 should be decided but s_2 itself is dependent on s_1. This case is most general one . . In terms of which any arbitrary computation can be organized. More precisely any arbitrary computation will simply consist of a nesting of computation of the kind in several levels.

Now we can see computational work in each case.

Case 1

One can represent this as follows

$$X_2(s) = \alpha(2,s) = \text{SQUARE ROOT OF } 2$$

Here once 's' is specified programme $X_2(s)$ can be executed to compute the value.

Case 2

This can be represented as

$$Z(s) = \alpha(X_2(s_1), X_3(s_1\ s)$$

Here 's' will be specified and programmes of $X_2(s_1)$ and $X_3(s_1)$ programmes are available. Let us see for s = 250,

$$Z(250) = X_2(s_1) * X_3(s_1)$$

where $s_1 = \min_{v \geq 1} v \left\{ \max \left| \alpha_1(v) * \alpha_2(v) - (\alpha_1(v) \pm \frac{1}{v}) * (\alpha_2(v) \right| \leq \frac{1}{250} \right\}$

For v = 1,2,3, ... we stop at when the error becomes less than $\frac{1}{250}$ and put $s_1 = v$.

Case 3

This can be represented as follows

$$Y(s) = Z_1(s_1) * Z_2(s_1)$$

where $s_1 = \min_{v \geq 1} v \{ \max \left| Z_1(v) * Z_2(v) - (Z_1(v) \pm \frac{1}{v}) * (Z_2(v) \pm \frac{1}{v}) \right| \leq \frac{1}{s} \}$

For a given value of s start with v = 1,2,3,4, ... and check the above condition on s_1 to be satisfied. That value of v gives s_1. Here $Z_1(v)$ for each value of v will be

$$Z_1(v) = X_1(s_2) * X_2(s_2)$$

where $s_2 = \min_{r \geq 1} r \{ \max \left| X_1(r) * X_2(r) - (x-r) \pm \frac{1}{r}) * (X_2(r) \pm \frac{1}{r}) \right| \leq \frac{1}{v} \}$

Again $X_1(r)$ for a given r may be given as follows

$$X_1(r) = X_{11}(s_5) * X_{12}(s_5)$$

where $s_5 = \min_{u \geq 1} u \{ \max \left| X_{11}(u) * X_{12}(u) - (X_{11}(u) \pm \frac{1}{u}) * (X_{12}(u) \pm \frac{1}{u}) \right| \leq \frac{1}{r} \}$

Evidently there is more computational work in case-3 than in case-2 or case-1.

This involved algorithmic approach can be made simple once we use modular programmes for case-1. Programmes of case-2 will use case-1 programmes. Programmes of case-3 type will use case - 1 and case-2 programmes. This modularity can be

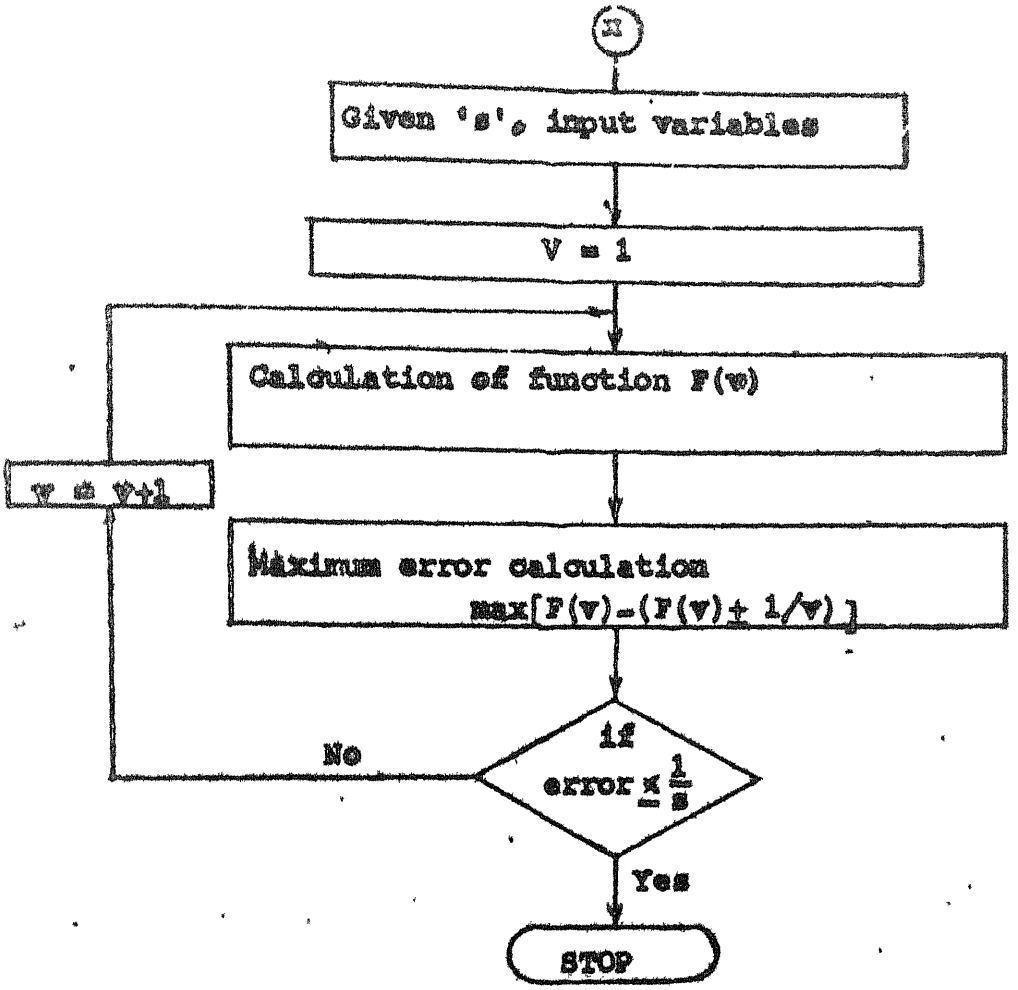


Fig. 5.2 Flow Chart of a Computable Number Programme.

Case 1 $F(v) = \alpha_1(v)$

Case 1 $F(v) = \beta_1(v)$

$= \alpha_1(u) \oplus \alpha_2(u)$

Case 3 $F(v) = \gamma_1(v)$

$= \beta_1(u) \oplus \beta_2(u)$

possible once we use the particular structure emerging from the flow chart of each case-1, case-2 and case-3. Using case-3 one can cover the full range of computable number programmes. This general structure is shown in figure 5.2.

5.2.7 Reduction in Computational Work

The next question is whether we can reduce computational work. Let us see what computational work is involved and then we will try to see if we can reduce it. Say we want to compute a computable number

$$Z(s) = X_2(s_1) * X_3(s_1)$$

where $s_1 = \min_{v \geq 1} v \{ \max | X_2(v) * X_3(v) - (X_2(v) \pm \frac{1}{v}) | \leq \frac{1}{s} \}$

The approach to implement it, for a given value of 's' may be as follows

1. start with v = 1
2. compute $X_2(v)$ and $X_3(v)$
3. compute max $|X_2(v) * X_3(v) - (X_2(v) \pm \frac{1}{v}) * (X_3(v) \pm \frac{1}{v})|$
4. if value from step 3 is less than or equal to $\frac{1}{s}$, stop execution else v = v+1 and jump to step 2.

Here v is incremented by 1 in every iteration. Thinking about step (2) of the algorithm, if

$$X_2(v) = Y_1(u) * Y_2(u)$$

where $u = \min_{r \geq 1} r \{ \max | Y_1(r) * Y_2(r) - (Y_1(r) \pm \frac{1}{r}) * (Y_2(r) \pm \frac{1}{r}) | \leq \frac{1}{v} \}$

Thus for each value of v, $X_2(v)$ has to be computed and incrementing v by 1. We have to reach point when $s_1 = v$. The computational work is thus proportional to the value of s_1.

To reduce the number of feedback loopings larger increments in v need to be considered, so an appropriate way can be to select a new value of v before going back for a new iteration, with the help of previously computed value in intermediate iterations.

Let us say

$$Z(s) = X_2(s_1) * X_3(s_1)$$

where $s_1 = \min_{v \geq 1} v \{ \max | X_2(v) * X_3(v) - (X_2(v) \pm \frac{1}{v}) * (X_3(v) \pm \frac{1}{v}) | < \frac{1}{s} \}$

Here $\max X_2(v) * X_3(v) - (X_2(v) \pm \frac{1}{v}) * (X_3(v) \pm \frac{1}{v}) = \frac{X_2(v)}{v} + \frac{X_3(v)}{v} + \frac{1}{v^2}$

So for new value v_n we can use old value v_1 and $X_2(v_o)$ and $X_3(v_o)$ as follows

$$\frac{1}{s} = \frac{X_2(v)}{v} + \frac{X_3(v)}{v} + \frac{1}{v^2}$$

or $v_n \quad (X_2(v_o) + X_3(v_o) + \frac{1}{v_o}) * s$

Here v_n denotes value of v for the next iteration in feedback. Proceeding in this manner can generally help cut down the number of feedbacks. We give below two examples of computations carried out in this manner.

First one is regarding computation of

$$Z(s) = \sum_{i=1}^{n} Y(i,1,s_1)*Y(i,2,s_1)$$

where $n = 1,2,3$

Second one is about the matrix inversion with computable analysis. Programmes used for both are listed in Appendix. In the result listed Tables (1) and (2), we see that there is a great reduction in number of iterations on using suggested computation reduction approach.

Table 1

Z	Terms	$Y(i,1,s_1)$	$Y(i,2,s_1)$	$Z(i,s_2) = Y(i,1,s_1) \cdot Y(i,2,s_1)$	$Z(100) = \Sigma Z(i,s_2)$	s_1	s_2	No. of iterations
(1) Using $s = s+1$ in iteration								
3 x 4	1	1.7326	2.0026	3.4698	3.4698	374	100	374
3 x 4 + 3 x $(5)^{\frac{1}{3}}$	2	1.7322	2.0013	3.4668	6.432	623	200	1436
		1.7329	1.7111	2.9653		683	200	
3 x 4 + 3 x $(5)^{1/3}$ + $(2)^{1/4}$ x 3	3	1.7321	2.0008	3.4658	8.490	1120	300	3031
		1.7321	1.7108	2.9633		1034	300	
		1.1892	1.7331	2.0612		877	300	
(2) Using $s_1 = (Y(i,1,s_1)+Y(i,2,s_1)+1.0) \times s_2$ in iteration								
3 x 4	1	1.7340	2.0020	3.471	3.471	500	133	2
3 x 4 + 3 x $(5)^{1/3}$	2	1.7330	2.0010	3.4677	6.430	1000	267	4
		1.7325	1.7100	2.9625		800	232	
3 x 4 + 3 x $(5)^{1/3}$ + $(2)^{1/4}$ x 3	3	1.7326	2.0006	3.4664	8.491	1500	401	6
		1.7325	1.7100	2.9625		1200	348	
		1.1900	1.7325	2.0616		1200	410	

Table 2

Matrix A			Matrix Inverse A		
3	1	2	0.3889	0.0555	-0.2778
2	3	1	-0.2778	0.3889	0.0555
1	2	3	0.0555	-0.2778	0.3889

Value of error in elements of Matrix A, $\frac{1}{s_1} = \frac{1}{367}$

Maximum error in elements of Matrix Inverse A, $1/s = 1/100$

Number of iterations (using $s_1 = s_1+1$) = 367

Number of iterations (using $s_1 = X(s_1)*s$) = 2

3	1	2	8	0.3222	-0.0111	-0.3044	0.0400
2	3	1	7	-0.3444	0.3222	0.0288	0.0400
1	2	3	9	-0.0777	-0.4111	0.3355	0.0800
9	8	7	6	0.0666	0.0666	0.0266	-0.0400

Value of error in elements of Matrix A, $\frac{1}{s_1} = \frac{1}{1267}$

Maximum error in elements of matrix inverse A, $\frac{1}{s} = \frac{1}{100}$ (given)

Number of iterations (using $s_1 = s_1+1$) = 1267

Number of iterations (using $s_1 = X(s_1)*s$) = 2

Here $X(s_1)$ is the value of largest element of corresponding matrix.

Section 6

Digital Signal Processing and Computable Numbers

6.1 Introduction

Computing machines are used in digital signal processing (DSP), both for the design of DSP systems e.g. digital filters and for the implementation of such systems.

The software approach is now used extensively. Numerical algorithms and programmes have now become basic tools used in digital signal processing. The existing methods of realization and design of DSP systems are all based on real analysis in the maximum accurate possible (MAP) form. As a result they pose difficulties that were discussed in section 3. On account of these difficulties, we felt that computable analysis would offer a more appropriate alternative as the computable number model specifically takes into account the problems of machine representation. Thus we can think of using computable numbers instead of real numbers in digital signal processing. More specifically.

1. System theoretic properties of DSP algorithms can be reformulated in terms of computable analysis.

2. Errors due to finite word length in a DSP algorithm can be looked upon in terms of computable analysis. In this manner the 'Dead band effect' and the sensitivity of DSP algorithms can perhaps be better understood.

3. Using the computable number concept can also be incorporated in the design of a DSP algorithms e.g. digital filters.

The third task can be carried out in terms of computable analysis only when the practical tools suggested in section 5 have been fully developed. So we will discuss only (1) and (2).

6.2 Basic Properties

First of all we shall look at some basic properties of digital filters to see how they can be rephrased. The properties to be discussed are :

1. Time invariance
2. Linearity
3. Causality
4. Stability

Basically a digital filter is a numerical algorithm have finite computations to be carried out using available values of the input sequence and the already computed values of

the output. Each arithematic operation we wish to consider in the computable number sense. Thus we find a digital filter algorithm is essentially a computable process; 'α' (section-4) which produces output $y(s)$ from input $x(s_1)$

$$y(s) = \alpha\ (x(s_1))$$

where

$$s_1 = \min_{v \geq 1} v \ \{\max \left| \alpha\ (x(v)) \ - \alpha\ (x(v) \pm \frac{1}{v}) \right| \leq \frac{1}{s}\ \}$$

Here s in $y(s)$ is the error parameter. The $x(s)$ and $y(s)$ have the computable number $x(n, s_1)$ and $y(n, s)$ respectively as its value at the nth point.

Time Invariance : Digital signal processing algorithm is said to be time invariant if

$$y(s) = \alpha\ (x(s_1))$$

implies $$y^{\tau}(s) = \alpha\ (x^{\tau}(s))$$

where $x^{\tau}(s_1)$ is the shifted sequence whose values are

$$x^{\tau}(n, s_1) = x(n-\tau, s_1)$$

for the nth element of $x^{\tau}(s_1)$.

Linearity : In terms of computable numbers, linearity of a digital signal processing algorithm would require that

$$(1)\ \alpha\ K(s_1)\ x\ (s) = K(s_1)\ \alpha\ X(s)$$

$$(2)\ \alpha\ X_1(s) + x_2(s) = \alpha\ x_1(s) + \alpha\ x_2(s)$$

where $x_1(s)$ and $x_2(s)$ are sequences of input and $K(s_1)$ is a computable number.

Causality : Digital signal processing system is causal if the computable input values are as follows.

$$x_1(n,s) = x_2(n,s_1) \text{ for } n \leq k$$

then $$y_1(n,s) = y_2(n,s_1) \text{ for } n \leq k$$

even for $$x_1(n,s) \neq x_2(n,s_1) \text{ for } n > k.$$

Here $x_1(n,s)$ and $x_2(n,s)$ are input valuesand $y_1(n,s)$ and $y_2(n,s_1)$ are the corresponding output values.

Stability : A digital signal processing algorithm is said to be stable if any bound input values results in a bounded output values i.e.

$$\text{if } x(n,s) < M_x(s_1)$$

for all values of n, then there is a computable number $M_y(s_2)$ such that

$$y(n,s) < M_y(s_2)$$

for all values of n. Here $M_x(s_1)$ is a computable number which is greater than input values for all n.

6.3 Finite word Length and Computable numbers :

Let us now examine the limitations resulting from finite word length from the point of view of computable numbers. Finite word length in the implementation of a digital hardware can produce three different type of errors. For the sake of concreteness let us take a specific case of a digital filter of first order which can be represented in real analysis as a difference equation.

$$y(n) = x(n) - a.y(n-1) \quad \ldots \qquad (6.1)$$

where y(n) and x(n) represents outputs and inputs of the digital filter. The difference equation when written in terms of computable numbers is as follows -

$$y(n,s) = x(n,s_1) - a\ (s_2).\ y(n-1),s_2) \qquad (6.2)$$

The three different types of errors are as follows :

1. A real number x(n) when represented on a machine will have the so called input quantization error.
2. The multiplier value 'a', will be represented with the multiplier coefficient quantization error.
3. In the rounding operation in multiplier, there exists rounding error.

Total error will be the error bound on the computable number y(n,s). Computable number concept also states that for

value of 's' once specified, there exists a method to compute $y(n,s)$. For this we require error computation on $y(n,s)$ and controlling it to a value less than $\frac{1}{s}$. Control of each of three error contributions can be achieved as follows :

1. To reduce input quantization error, decrease the step of quantization.
2. To reduce multiplier coefficient quantization error, one can represent multipliers with greater precision,
3. To reduce rounding error one can use longer word length

For the same of simplicity we assume that each type of error can be studied independently of the others.

6.3.1 Input quantization error:

Let us take $\Delta s = p$ as the step size for the quantization of input and digital filter represented by difference equation (6.2) starts operation at $n = 0$ with initial conditions as follows.

$$x(-1) = x(-2) = \ldots = 0$$

$$y(-1) = y(-2) = \ldots = 0$$

So equation

$$y(n,s) = x(n,s_1) - a(s_2).y(n-1,s_2)$$

can be written for $n = 0$ as

$$y(0,s_1) = x(0,s_2) - a.0$$

(since $a(s_2) = a$ and no rounding error assumed)

where

$$\frac{1}{s_1} = \frac{1}{s_2} = \frac{p}{2}$$

For $n = 1$

$$y(1,s_3) = r(1,s_2) - a.y(0,s_1)$$

where

$$\frac{1}{s_3} = \frac{1}{s_2} + \frac{a}{s_1}$$

$$= \frac{p}{2} + \frac{ap}{2} = (1+a)\frac{p}{2}$$

For $n = k$

$$y(,s_{k+1}) \; x(k,s_2). \; a.y(k-1,s_k)$$

where

$$\frac{1}{s_{k+1}} = \frac{1}{s_2} + \frac{a}{s_2}(1 + a + \ldots + a^{k-2})$$

$$= \frac{1}{s_2}(1+a+a^2+ \ldots + a^{k-1})$$

$$= \frac{p}{2}\left(\frac{1-a^k}{1-a}\right), \quad \text{for } a<1 \quad \ldots. \qquad (6.3)$$

Thus the error contribution in the output at n=k is given by equation 6.3 . This indicates that after starting operation at n = 0, error is accumulating but once k is large, $a^n << 1$ as a is less than unity, in general.

$$\frac{1}{s_{k+1}} \simeq -\frac{p}{2} \cdot \frac{1}{1-a}$$

Now let us suppose at n = k+1 input is removed.

$$y(k+1, s_{k+2}) = - a.y(k, s_{k+1})$$

where

$$\frac{1}{s_{k+2}} = a \cdot \frac{1}{s_{k+1}}$$

$$= a. \frac{p}{2} \left(\frac{1-a^k}{1-a}\right)$$

Also

$$Y \quad y(k+2, s_{k+3}) = -a.y(k+1, s_{k+2})$$

where

$$\frac{1}{s_{k+3}} = a^2 \cdot \frac{p}{2}\left(\frac{1-a^k}{1-a}\right)$$

Thus

$$y(k+m, s_{k+1+m}) = -a.y\ (k+m-1, s_{k+m})$$

where

$$\frac{1}{s_{k+1+m}} = a^m \cdot \frac{p}{2} \left(\frac{1-a^k}{1-a}\right)$$

For $a < 1$, $\frac{1}{s_{k+2}}$ is less than $\frac{1}{s_{k+1}}$ and $\frac{1}{s_{k+1+m}}$ is still lesser. This means that the quantization error contribution decays after the input samples are made zero.

6.3.2 Multiplier coefficient Quantization error and Sensitivity:

This error is contributed by approximate representation of a multiplier. Errors due to rounding in multiplier and input quantization is taken to be zero. Starting computation of $y(n,s)$ using equation 6.2.

At $n = 0$, initial conditions are as follows.

$$x(-1) = x(-2) = x(-3) = \ldots = 0$$

$$y(-1) = y(-2) = y(-3) = \ldots = 0$$

Then equation 6.2 at $n = 0$ reduces to the form

$$y(0,s_o) = x(0) - a(s_2).0$$

where $\frac{1}{s_o} = 0$; $(x(0,s) = x(0)$ as assumed)

At $n=1$

$$y(1,s_3) = x(1) - a(s_2).\, y(0,\, s_o)$$

where $$\frac{1}{s_3} = \max \left| a(s_2).y(o,s_o) - (a(s_2) \pm \frac{1}{s_2}).(y_o,s_o) \pm \frac{1}{s_o}) \right|$$

$$= \frac{y(o,s_o)}{s_2} = \frac{x(0)}{s_2}$$

At n =2

$$y(2,s_4) = x(2) - a(s_2).y(1,s_3)$$

where $$\frac{1}{s_4} = \max \left| a(s_2).y(1,s_3) - (a(s_2) \pm \frac{1}{s_2}).(y(1,s_3) \pm \frac{1}{s_3}) \right|$$

$$= \frac{1}{s_2}.y(1,s_3) + \frac{a(s_2)}{s_3} + \frac{1}{s_2 s_3}$$

$$= \frac{x(1) - a(s_2).x(0)}{s_2} + a(s_2).\frac{x(0)}{s_2} + \frac{x(0)}{s_2^2}$$

$$= \frac{x(1)}{s_2} + \frac{x(0)}{s_2} + \frac{2\,x(0)a(s_2)}{s_2}$$

(taking maximum value)

$$= \frac{x(1)}{s_2} + x(0)\left((a(s_2) + \frac{1}{s_2})^2 - a^2(s_2)\right)$$

. . .

For n = k

$$y(k, s_{k+2}) = x(k) - a(s_2).y(k-1, s_{k+1})$$

where $$\frac{1}{s_{k+2}} = \sum_{m=0}^{k-1} x(m)\,(a(s_2) + \frac{1}{s_2})^{k-m} - a^{k-m}(s_2) \qquad .. (6.4)$$

This indicates that output error increases as the value of n increases. Another form of difference equation can be taken as follows .

$$y(n,s) = b_1(s_3).x(n,s_1) - a(s_2) - y(n-1,s_2)$$

In this case now we can see output error contributed by error $\frac{1}{s_3}$ in representation of $b_1(s_3)$. Initial conditions are given at n=1 as follows.

$$x(-1) = x(-2) = \ldots = 0$$

$$y(-1,s) = y(-2,s) = \ldots = 0$$

Here input quantization erro rounding erro and error due to $\frac{1}{s_2}$ with $a(s_2)$ is taken to be absent.

At $n = 0$

$$y(o,s_o) = b_1(s_3)x(0) - a.y(-1,s_2)$$

$$= b_1(s_3).x(0)$$

(Here $\frac{1}{s_1}=0$ assumed)

where $$\frac{1}{s_o} = \frac{x(0)}{s_3}$$

At $n = 1$

$$y(1,s_4) = b_1(s_3).x(1)-a.y(o,s_o)$$

$$= b_1(s_3).x(1)-a.b_1(s_3).x(0)$$

where $$\frac{1}{s_4} = \frac{x(1)}{s_3} + \frac{a}{s_o}$$

$$= \frac{x(1)}{s_3} + \frac{a.x(0)}{s_3}$$

At $n = 2$

$$y(2, s_5) = b_1(s_3)x(2) - a.y(1, s_4)$$

where

$$\frac{1}{s_5} = \frac{x(2)}{s_2} + \frac{a}{s_4}$$

$$= \frac{x(2)}{s_3} + \frac{a.x(1)}{s_3} + \frac{a^2.x(0)}{s_3}$$

Thus for $n = k$

$$y(k, s_{k+3}) = b_1(s_3).x(k) - a.y(k-1, s_{k+2})$$

where $$\frac{1}{s_{k+3}} = \frac{1}{s_3}\left(x(k-1) + a.x(k-2) + \ldots + a^k.x(0)\right)$$

$$= \frac{1}{s_3} \sum_{n=0}^{k-1} x(m)a^{k-m} \qquad (6.5)$$

Here too, the error contribution increases with increase in n. This error can be reduced by reducing error bound $1/s_3$ on $b_1(s_3)$ and $1/s_2$ on $a(s_2)$.

Sensitivity : A concept related to this error contribution is the sensitivity of a digital filter structure. Sensitivity of a structure is defined as the ratio of error contribution in output due to change in mutliplier coefficient to change in multiplier coefficient. Structures taken are direct, cascade

and parallel form and each stage is restricted to first order only whose difference equation representation is given by equation 6.2 .

1. Direct Form : Let a direct form of a digital filter as shown in Fig. 6.1(a) be

$$y(n,s_1) = a_o(s_{o1})\,x(n) + a_1(s_{11})\,x(n-1) + \ldots + a_N(s_{N1}) \cdot x(n-N) \quad \ldots (6.6)$$

Let us take $a_o(s_{o1})$ multiplier value that have $\frac{1}{s_{o1}}$ error bound. All other multipliers are assumed to have no multiplier quantization error. Thus we can rewrite equation (6.6).

$$y(n,s_1 = a_o(s_{o1})x(n) + a_1.x(n-1) + \ldots + a_N x(n-N)$$

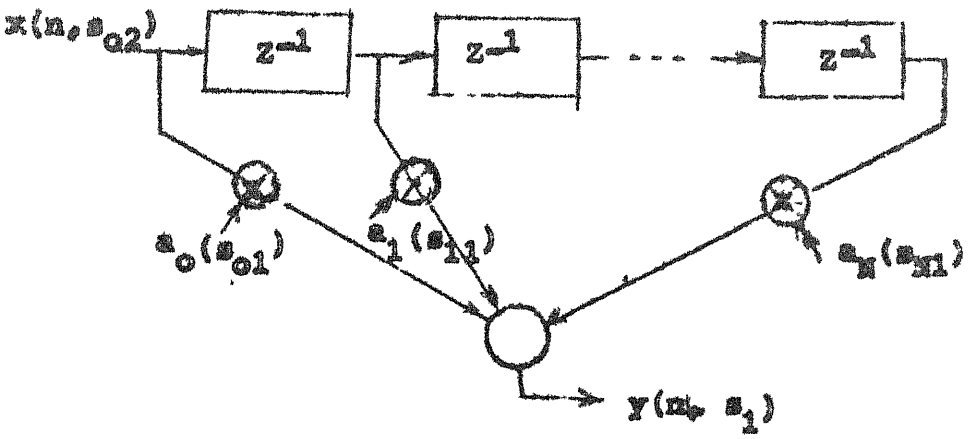
where $$\frac{1}{s_1} = \frac{x(n)}{s_{o1}}$$

Thus sensitivity $$= \frac{1/s_1}{1/s_{o1}} = x(n) \quad \text{--- (6.7)}$$

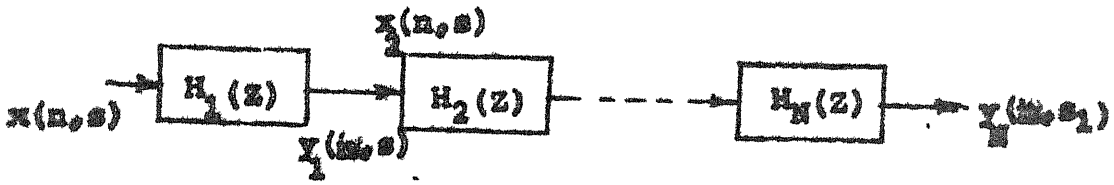
2. Cascade Form : This form of structure is shown in figure 6.1(b). All the stages are put in cascade. Let us represent i^{th} stage as follows

$$y_i(n,s) = b_1(s_1)x(n) - a_i(s_2)y(n-1,s_3).$$

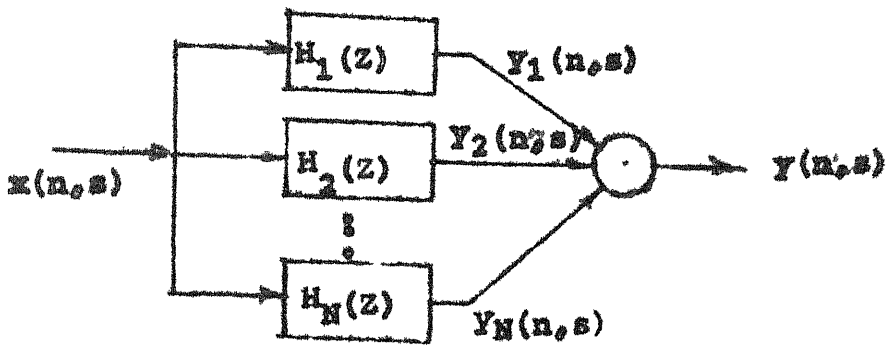
where $i = 1, 2, \ldots N.$



(a) Direct Form



(b) Cascade Form



(c) Paralle Form

Fig. 6.1 Digital Filter Structures

At first let $b_i(s_1)$ have $1/s_1$ error bound and $1/s_2 = 0$ on a $a(s_2)$.

Here $$y_i(s) = x_{i+1}(s)$$

or $$\Delta y_i(n,s) = \Delta x_{i+1}(n,s) = \frac{1}{s_1} \left(x_i(n,s) + a_i(s_2) x_i(n-1,s) + \ldots + a_i^n(s_2) x_i(o,s) \right) \qquad (6.8)$$

from equation (6.5)

This is now error in input $x_{i+1}(n,s)$ being denoted as E_n. Thus output error

$$\Delta y_{i+1}(n,s) = b_{i+1} (E_n + a_{i+1} E_{n-1} + \ldots + a_{i+1}^{n-1} E_1)$$

This is again an input error to $i+2^{nd}$ stage and so on. This may any error from first stage gets enlarged by successive stages, in other words front stages are more sensitive than the latter ones.

Now for second case, let $a_i(s_2)$ have $\pm \frac{1}{s_2}$ error bound but $1/s_1 = 0$ on $b_i(s_1)$. Then using equation 6.4 we have

$$\Delta y_i(n,s) = E_n = \sum_{m=o}^{n-1} x_m (s_1 \; a_i(s_2) + \frac{1}{s_2})^{n-m} - a_i^{n-m}(s_2) \qquad \ldots \quad (6.9)$$

Here $y_i(n,1) = x_{i+1}(n,s)$

or $x_{i+1}(n,s)$ has $\pm E_n$ as input error, which will result $y_{i+1}(n,s)$

$$\Delta y_{i+1}(n,s) = b_{i+1}(s_1)\,(E_n + a_{i+1}(s_2)\, E_{n-1} + \ldots + a_{i+1}^{n-1}(s_2)\, E_1)$$

Thus if a multiplier coefficient varies in first stage, output variation will be larger than a case when a multiplier coefficient vary in a latter stage.

3. Parallel Form : This form is shown in Fig. 6.1(c). All stages are put in paralle. Let us represent this structure as follows

$$y(n,s) = \sum_{i=1}^{N} y_i(n,s_3)$$

and $y_i(n,s_3) = b_i(s_1)x(n,s_1) - a_i(s_2)y_i(n-1,s_2)$

If we take the case when $1/s_1$ error on $b_i(s_1)$ exist alone.

$$\Delta y(n,s) = \Delta y_i(n,s_3) = \frac{1}{s_1}(x(n,s) + a_i(s_2)x(n-1,s) + \ldots + a_i^n(s_2)x(o,s))$$

from equation (6.5)

Similarly if we have only $a_i(s_2)$ with error $\frac{1}{s_2}$,

$$\Delta y(n,s) = \Delta y_i(n,s_3) = \sum_{m=o}^{n-1} x(m,s)\ \ (a_i(s_2) + \frac{1}{s_2})^{n-m} - a_i^{n-m}(s_2)$$

from equation (6.4).

Thus here cascading effect is not there as was in cascade form. This may cascade form contributes more error at output due to a variation in multiplier coefficient of any stage except last one, than the parallel form. Direct form has minimum sensitivity in these three forms. Parallel form is less sensitive than cascade form.

6.3.3 Rounding error of Multipliers and Dead band Effect

This error contributed in output is by multiplication of two numbers represented in b_1 bits each rounded to b_1 bits only and not $2b_1$ bits. Filter is represented by equation (6.2)

$$y(n,s) = x(n,s_1) - a(s_2).y(n-1,s_2)$$

Here errors due to input quantization and multiplier coefficient quantization are assumed to be absent. Say multiplication $a(s_2).y(n-1,s_2)$ has $\pm$ p/2 error due to rounding.

Starting conditions at n=o are as follows

$$x(-1) = x(-2) = x(-3) = \ldots = 0$$
$$y(-1) = y(-2) = y(-3) = \ldots = 0$$

Then equation (6.2) at n = 0 reduces to the form

$$y(o,s_o) = x(o,s_1) - a(s_2).0$$

where $\frac{1}{s_o} = 0$ ($\frac{1}{s_1} = 0$ as assumed)

Here $x(o,s_1) = x(0)$ and $a(s_2) = a$

At $n = 1$

$$y(1,s_3) = x(1) + a.y(o,s_o)$$

where $1/s_3 = 0 + p/2$

At $n = 2$

$$y(2,s_4) = x(2) - a.y(1,s_3)$$

where $1/s_4 = 0 + a/s_3 + p/2$

$$= p/2(1+a)$$

At $n = 3$

$$y(3,s_5) = x(3) - a.y(2,s_4)$$

where $1/s_5 = 0 + a/s_4 + p/2$

$$= p/2 \quad (1+a+a^2)$$

Thus at $n = k$

$$y(k,s_{k+2}\Sigma = x(k) - a.y(k-1,s_{k+1})$$

where $1/s_{k+2} = p/2 \quad (1+a+ \ldots +a^{k-1})$

$$= p/2 \cdot \frac{1-a^k}{1-a}$$

For a large value of k and a <1

$$1/s_{k+2} \simeq p/2 \cdot \frac{1}{1-a} \qquad (6.10)$$

So error contributed at output will remain at $p/2 \cdot \frac{1}{1-a}$ value after a large value of n.

Now we will see, if at n=k we remove input, then how this erro contribution is going to change.

At n = k+1, input is removed.

$$y(k+1, s_{k+3}) = - a \cdot y(k, s_{k+2}) \qquad -- \quad (6.11)$$

where $$\frac{1}{s_{k+3}} = \frac{p}{2} + \frac{a}{s_{k+2}}$$

$$= p/2 + p/2\ a \cdot (1+a+\ldots+ a^{k-1})$$

$$= p/2\ (1 + a + \ldots + a^{k})$$

At n = k+2

$$y(k+2, s_{k+4}) = - a \cdot y(k, s_{k+3})$$

where $$\frac{1}{s_{k+4}} = p/2 + a/s_{k+3}$$

$$= p/2\ (1+a+a^{2} +\ldots+a^{k})$$

Thus at n = k+m

$$y(k+m, s_{k+m+2}) = - a \cdot y(k+m-1, s_{k+m+1})$$

where $\frac{1}{s_{k+m+2}} = p/2 + \frac{a}{s_{k+m+1}}$.

$$= p/2\ (1+a+ \ldots + a^{k+m-1})$$

$$= p/2\ \frac{1-a^{n+m}}{1-a}$$

So for n taken larger than 10 and $a < 1$

$$\frac{1}{s_{k+m+2}} \simeq p/2\ .\ \frac{1}{1-a} \qquad (6.12)$$

Thus $1/s_{k+2} \simeq 1/s_{k+m+2}$ and this error is maintained though output will decay as input has been removed. By equation (6.11) we see that $y(k, s_{k+2})$ is multiplied by a value less than 1 so we can write

$$y(k, s_{k+2}) > y(k+1, s_{k+3}) > y(k+2, s_{k+4}) \text{ etc.}$$

In Digital Signal Processing theory, this is termed as the Dead band effect, that is even if input is zero, output will oscillate between between p/2 . 1/1-a to -p/2.1/1-a. Let us now compute dead band effect bound using the conventional real analysis.

<u>The Dead-Band effect</u> : We generally represent a digital filter by a difference equation of the form, in real analysis as follows

$$y(n) = \sum_{k=0}^{n} b_k x(k) - \sum_{1=0}^{n-1} a_1 . y(1)$$

Let us have following conditions :

1. initially x(k) and y(1) are zero so y(n) = 0.
2. At n=0 input was non zero and one gets output also non zero.
3. After a long time input is removed at $n \geq n_1$.

In such a situation the output y(n) for $n \geq n_1$ will also decay toward zero if system is stables and real analysis is used.

In practice due to finite word length of the machine one will have rounding errors accompanying the output.

For example for first order difference equation

$$y(n) = x(n) - 0.8\, y(n-1)$$

find the nature of y(n), $n \geq 0$

Given $y(-1) = 8$, $x(k) = 0$ for $k > 0$; Result in tabular form is :

n	x(n)	y(n-1)	-ay(n-1)	y(m)
0	0	8	-6.4	-6
1	0	-6	+4.8	5
2	0	5	-4.0	-4
3	0	-4	3.2	3
4	0	3	-2.4	-2
5	0	-2	1.6	2
6	0	2	-1.6	-2

Here we carryout rounding to the nearest integer at each computation. The Dead band effect is visible after n=4 in the table given.

A general technique to study the dead band effect was given by Jackson [10,P296] Let us briefly discuss this.

It is assumed that at each product a.y(n-1) in

$$y(n) = x(n) - a.y(n-1) \quad .. \quad (6.13)$$

is rounded to the nearest integer. According to this rule

$$\text{Qnt.}\ [\,|a.y(n-1)|\,] = \text{Integer}\ [\,|a.y(n-1)| + 0.5\,] \quad .. \quad (6.14)$$

When input is absent,

$$y(n) = -a.y(n-1)$$

The condition for y(n) and y(n-1) to be equal is that

$$\text{Qnt.}\ [\,|a.y(n-1)|\,] = |y(n)| = |y(n-1)|$$

$$\text{or Int.}\ [\,|a.y(n-1)| + .5\,] = y(n-1) \quad .. \quad (6.15)$$

from rule (6.14)

$$\text{or Int.}\ [\,|y(n-1)| - (1-|a|)\,|y(n-1)| + 0.5\,] = |y(n-1)|$$

This means that

$$0 < -(1-|a|)\,|y(n-1)| + 0.5 < 1$$

$$\text{or} \quad |y(n-1)| \leq \frac{0.5}{1 - |a|} \qquad \ldots (6.16)$$

As $y(n-1)$ can have only integer values for $|a| < 0.5$; $y(n-1) = 0$. Thus there is no dead band effect.

Comparing equation (6.12) and equation (6.16) we find that computable analysis provide an explanation of the dead band effect. Now let us see whether the absence of dead band effect for a < .5 can also be explained using the concept of computable analysis. Answer is that for a < 0.5, the rounding error will not be there in such cases, thus so rounding error will also decay as the value of y(n,s).

Section 7

Conclusion

The aim of this thesis has been to present the idea of computable number and the basic concept of computable analysis to those who are concerned about D S P. Originally it had been envisaged that whatever the results on computable analysis were already available, they could be used to reformulate the theory of digital signal processing in a straight forward manner. It however turned out that the task was not as straight forward and that a considerable amount of background needed to develop, before it could be seen how computable analysis could be incorporated into Digital Signal Processing. The results of the efforts made in this direction are what is contained in this thesis.

Among the major points that seem to emerge from our study are the following :

1. A more satisfactory theory of Digital Signal Processing can be developed if we abandon real analysis. We use a kind of theory of numerical analysis that makes use of the theory of recursive functions and idealized computer model such as URM.

2. In practice a lot of Digital Signal Processing is carried out using dedicated hardware that has severe limitation of word length. The computable number analysis in this context provides a more appropriate mathematical framework within which signal processing algorithms can be studied.

3. For the theory of computable analysis to become practically usable, a great deed of work needs to be done to rewrite library functions and procedures and subroutines for arithematic operations and relations in the language of computable analysis.

Special direction for future work as the followings :

1. Complete reformulation of the theory of digital signal processing can be attempted, for which a start has been made in this thesis.

2. Once we accept the idea of computable number are a programme, an explicit algebra of programmes needs to be evolved. In this algebra given a programme we should talk of inverse programme and so on. The purpose of such an algebra will be to simplify the task of implementing computable analysis. Thus a computable number $\alpha(s)$ is given in form of a programme we should be able to cuickly construct the inverse programme corresponding to $1/\alpha(s)$.

References

1. Aberth, O., 'Computable Analysis' McGraw-Hill, New York, 1980.

2. Cutland, N.J., 'Computability An Introduction to Recursive Function Theory'., Cambridge University Press, Cambridge, 1980.

3. Myhill, J., 'What is a real numbers?' Americal Mathematical monthly, vol. 79 No.9 p 748-754, Sept. - 1972.

4. Bishop, E., 'Foundation of Constructive Analysis' McGraw Hill, New York, 1967.

5. Brigdes, D.S., 'Constructive Functional Analysis' Pitsman Publishing Limited, London 1979.

6. Moore, R.R., 'Interval Analysis', Prentice Hall, Inc., N. Jersey, 1966.

7. Alefeld, G., Grigorieff, R.D., 'Foundation of Numerical Computations', Springer-Verlad, Wien, New York, 1980.

8. Anderson, J.A., 'Real Analysis', Logos Press Limited, London, 1969.

9. Leren, U., 'Structuring Mathematical Proofs', American Mathematical Monthly, vol. 90 No.3, P 174-185, March 1983.

10. Antoniou,A., 'Digital Filters - Analysis and Design' Tata McGraw Hill New Delhi, TMH edition, 1980.

11. Dedetind,R., 'Essays on the Theory of Numbers', Dovers Publications Inc. New York, 1963.

12. Parkar,F.D., 'The Structure of Number Systems' Prentice-Hall Inc., N. Jersey, 1966.

13. Drobot,S., 'Real Numbers', Prentice Hall Inc. New Jersey, 1964.

14. Kafoury,A.J., 'A Programming Approach to Computability' Springer-Verlag, N. York, 1982.

APPENDIX

```
C       GETTING SQUARE ROOT WITH 1/N ACCURACY
C       AND USING IT AS A COMPUTABLE NUMBER IN COMPUTABLE ANALYSIS
C       BY PRITHVI SINGH PUROHIT EE/8310421
        DIMENSION X(5,5),Y(5,5),Z(5)
        OPEN(UNIT=5,DEVICE='DSK',FILE='POUT.FOR')
        OPEN(UNIT=4,DEVICE='DSK',FILE='PIN.FOR')
        READ(4,*)NTRM,ERBND
        CALL ADSUB(NTRM,ERBND,ERBND1)
        ZT=0.0
        WRITE(5,2002)
 2002   FORMAT(5X,'Y(I,1)',10X,'Y(I,2)',5X,'Y(I,1)*Y(I,2)',10X,'S1',10X
       1,'S2')
        INBR=0
        DO 100 I=1,NTRM
        READ(4,*)(X(I,J),J=1,4)
        S1=1.0
  102   CALL SQRTT(Y(I,1),X(I,1),X(I,2),S1)
        CALL SQRTT(Y(I,2),X(I,3),X(I,4),S1)
        CALL MLTPY(Y(I,1),Y(I,2),S1,S2,Z(I))
        INBR=INBR+1
        IF(1.0/S2-ERBND1)121,121,101
  101   S1=(Y(I,1)+Y(I,2))/ERBND1
C 101   S1=S1+1.0
        IF(S1-29800.0)102,102,121
  121   ZT=ZT+Z(I)
        WRITE(5,*)Y(I,1),Y(I,2),Z(I),S1,S2
  100   CONTINUE
        WRITE(5,2005)ZT,ERBND,NTRM,INBR
 2005   FORMAT(10X,'TOTAL VALUE',F12.3,/,10X,'ERR. BOUND',F12.3,/,10X,'T
       1ERMS',I5,/,10X,'FEEDBACK LOOP',I6)
        STOP
        END
C       SUBROUTINE TO CALCULATE (X)**1/N
C       A0=OUTPUT  A1=STEP OF ROUNDING (2*S)  AB=POWER(VALUE OF N)
        SUBROUTINE SQRTT(A0,AVALU,AB,A1)
        A2=0.0
   11   A3=A2/A1
        A4=A3**AB
        A5=A4-AVALU
        IF(A5)16,16,15
   15   A0=A3
        RETURN
   16   A2=A2+1.0
        GO TO 11
        END
C       SUBROUTINE MULTIPLY  EI(BN)=X11(AN)*X22(AN)
C       ONLY TO CALCULATE ERROR BOUND "BN"
        SUBROUTINE MLTPY(X11,X22,AN,BN,EI)
        FF=1.0/AN
        EI=X11*X22
        ERTT=FF*(X11+X22)
        BN=1.0/ERTT
        RETURN
        END
C       SUBROUTINE DIVIDE FI(CN)=X33(CN)/X44(CN)
C       ONLY TO CALCULATE ERROR BOUND "DN"
        SUBROUTINE DVDE(X33,X44,CN,DN,FI)
        GF=1.0/CN
        FI=X33/X44
        EROR=GF*(1.0/X33+1.0/X44)
        DN=1.0/EROR
        RETURN
        END
C       SUBROUTINE ADDITION/SUBTRACTION HI(EN)=X55(EK)+/-X66(EK)+/-.....
C       ONLY TO CALCULATE ERROR BOUNB "EK"
        SUBROUTINE ADSUB(N,EN,EK)
        EK=EN/FLOAT(N)
        RETURN
        END
C       CALCULATION OF SIN(X(S)) WHICH HAS +/-(1/S) ERROR BOUND
C       BY PRITHVI SINGH PUROHIT
        SUBROUTINE SINE(X,S1,S2,SNE)
        EN=1.0/S2
        SNE=X
        N=1
        SNX=SIN(X)
        T2=X
        T3=X+EN
        SIXE=SIN(T3)
        SNER=X+EN
  202   N=N+1
        T1=T2
        T2=-T1*X*X/(FLOAT(2*N-1)*FLOAT(2*N-2))
        T3=-T3*(X+EN)*(X+EN)/(FLOAT(2*N-1)*FLOAT(2*N-2))
        SNE=SNE+T2
        SNER=SNER+T3
        R1=T2/T1
        EN1=T2*R1/(1.0-R1)
```

```
      ETT=ABS(EN1)+ABS(EN2)
      IF(ABS(EN1)-ABS(EN2))201,205,205
  205 STT=1.0/ETT
      IF(S2-STT)200,200,201
  201 IF(N-20000)202,202,200
C 200 WRITE(5,2001)SNE,SNER,N,ETT,EN1,EN2
C     WRITE(5,2000)SNX,SIXE,X,S1,S2,STT
 2001 FORMAT(2F12.6,I12,3F12.7)
 2000 FORMAT(2F12.6,4F12.3,//)
  200 RETURN
      END
```

```
C       PRITHVI SINGH PUROHIT EE/8310421
C       PROGRAM TO OBTAIN INVERSE OF A MATRIX
        DIMENSION B(10,10),A(10,20),AI(10,10),ER(10,20)
        OPEN(UNIT=5,DEVICE='DSK',FILE='MOUT.FOR')
        OPEN(UNIT=4,DEVICE='DSK',FILE='MIN.FOR')
        READ(4,*)N,S
        S1=1.0
        NCNTR=0
        DO 30 I=1,N
 30     READ(4,*)(B(I,J),J=1,N)
        WRITE(5,1010)
 1010   FORMAT(10X,'MATRIX "A"')
        DO 42 I=1,N
 42     WRITE(5,*)(B(I,J),J=1,N)
        M=N+N
        M2=N+1
 41     NCNTR=NCNTR+1
        DO 31 LI=1,N
        DO 44 LC=1,N
 44     A(LI,LC)=B(LI,LC)
        DO 31 LJ=M2,M
        ER(LI,LJ)=0.0
 31     A(LI,LJ)=0.0
        DO 32 K=1,N
        I2=K+N
 32     A(K,I2)=1.0
        ERR=0.0
        DO 33 LJ=1,N
        J2=LJ+1
        P=A(LJ,LJ)
        DO 34 I=1,M
C       ER(LJ,I)=(ABS(A(LJ,I))+1.0/S1)/(ABS(P)-1.0/S1)-ABS(A(LJ,I)/P)
 34     A(LJ,I)=A(LJ,I)/P
        DO 33 LK=1,N
        DO 33 LI=J2,M
        IF(LK-LJ)35,33,35
 35     ER(LK,LI)=(1.0+ABS(A(LK,LJ))+ABS(A(LJ,LI)))/S1
        A(LK,LI)=A(LK,LI)-A(LJ,LI)*A(LK,LJ)
        IF(ERR-ER(LK,LI))38,33,33
 38     ERR=ER(LK,LI)
 33     CONTINUE
        IF(ERR-1.0/S)39,39,40
C 40    S1=S*S1*ERR
 40     S1=S1+1.0
        IF(S1-10000.0)41,41,39
 39     DO 36 I=1,N
        DO 36 J=M2,M
        L3=J-N
        ER(I,L3)=ER(I,J)
 36     AI(I,L3)=A(I,J)
        WRITE(5,3003)
 3003   FORMAT(/,/,10X,'INVERSE "A" MATRIX')
        DO 37 I=1,N
 37     WRITE(5,*)(AI(I,J),J=1,N)
        WRITE(5,3000)S1,S,NCNTR
 3000   FORMAT(5X,'VALUE OF ERROR IN ELEMENTS OF MATRIX "A"',F10.2,/,
       1 5X,'MAXI. ERROR IN ELEMENTS OF MATRIX "INVERSE A"',F10.2,/,
       1 5X,'FEEDBACK LOOPS',I6,/,/,10X,'ERROR MATRIX OF INVERSE "A"')
        DO 88 I=1,N
 88     WRITE(5,*)(ER(I,J),J=1,N)
        STOP
        END
```

A 91867

EE-1985-M-PUR-COM.